Data-driven River Flow Routing using Deep Learning:

Predicting Flow along the Lower Orange River, Southern Africa

BY

C.J. Briers

Thesis presented in partial fulfilment of the requirements
for the degree of Master of Science (Applied Mathematics) in the Faculty of Science
at Stellenbosch University

Supervisor: Dr W.H. Brink
Co-supervisor: Prof. G.J.F. Smit

April 2019

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Abstract

The Vanderkloof Dam, located on the Orange River, is responsible for the water supply to consumers along its 1 400 km reach up to where it flows into the Atlantic Ocean. The Vaal River, which joins the Orange River approximately 200 km downstream of the dam, contributes significant volumes of water to the flow in the Orange River. These contributions are, however, not taken into account when planning for releases from the Vanderkloof Dam. In this thesis we aimed to develop an accurate and robust flow routing model of the Orange and Vaal River system to predict the effects of releases from the Vanderkloof Dam and anticipate inflows from the Vaal River. Since the factors that impact on flow rate and volume along the river are hard to quantify over long distances, a data-driven approach is followed which uses machine learning to predict the flow rate at downstream flow gauging stations based on flow rates recorded at upstream gauging stations. We restrict the model input to data that would be readily available in an operational setting, making the model practically implementable.

A variety of neural network architectures, including fully-connected networks, convolutional neural networks (CNNs) and recurrent neural networks (RNNs), were investigated. It was found that fully-connected networks produce results with accuracy comparable to a simple linear regression model, but display a superior ability to predict the timing of peaks and troughs in flow rate trends. CNNs and RNNs displayed the same ability, as well as showing improvements in accuracy. The best-performing CNN model had a mean absolute percentage error (MAPE) of 14.5 % compared to 16.9 % of a linear regression model.

To anticipate contributions from the Vaal River we investigated including inflows recorded at stations on the Vaal River and two of its tributaries, the Modder and Riet Rivers. Both approaches which were investigated, i.e. incorporating these inflows as part of multi-dimensional input into a CNN, and using a parallel CNN model architecture, showed promise with a MAPE of 21.6 % and 23.5 %, respectively. Although these models did not achieve a high level of accuracy, they did display the ability to anticipate contributions from the Vaal River system. It is believed that they could, with additional refinement or using appropriate safety factors, be practically applied in an operational setting.

We further investigated including seasonal data as input into our models. Including the time of the year, and including evaporation data recorded at meteorological stations in the recent past, both resulted in improved MAPE accuracy (14.4 % and 14.8 %, respectively, compared to 18.4 % for a model including no seasonal data). Observations of errors staying relatively constant over time prompted us to include errors made in the recent past as input into subsequent predictions. A model trained with this additional data achieved a MAPE of 10.2 %, a significant improvement over other applied methods.

Uittreksel

Die Vanderkloof-dam, wat op die Oranjerivier geleë is, verskaf water aan verbuikers langs die 1 400 km stroom-af rivierloop tot by die Atlantiese Oseaan. Die Vaalrivier, wat met die Oranjerivier ongeveer 200 km stroom-af van die dam saamvloei, maak 'n beduidende bydrae tot die vloei in die Oranjerivier. Dit word egter nie in ag geneem wanneer loslate uit die Vanderkloof-dam beplan word nie. In hierdie tesis beoog ons om 'n akkurate en prakties implimenteerbare vloei-roeterings model om die stroom-af effekte van loslate by die Vanderkloof-dam, en wat die bydraes van die Vaalrivier in ag neem, te ontwikkel. Aangesien faktore wat stroomvloei affekteer moeilik is om oor lang afstande te kwantifiseer, word data-gedrewe masjienleer-tegnieke toegepas deur vloeitempo wat by stroom-op stasies gemeet word te gebruik om vloeitempo by stroom-af stasies te voorspel. Om te verseker dat ons modelle in die praktyk aangewend kan word, beperk ons die invoer tot data wat in 'n operasionele omgewing beskikbaar is.

Implimentasie van vol-konneksie, konvolusionele en herhaal-terugvoer neurale netwerke was ondersoek. Volle-konneksie netwerke se resultate was vergelykbaar met dié van 'n lineêre regressie model, maar het die tydsberekening van stygings en daling in vloeitempo beter voorspel. Konvolusionele en herhaal-terugvoer netwerke het die tydsberekening goed voorspel, asook 'n verbetering in akkuraatheid getoon. Die model met die beste resultate was 'n konvolusionele netwerk met 'n absoluut gemiddelde persentasie-fout van 14.5 %, in vergelyking met 16.9 % vir 'n lineêre regressie model.

Om bydraes tot vloei vanaf die Vaalrivier in te sluit, is daar ondersoek ingestel om vloeimetings van meetstasies op die Vaalrivier en twee van sy sytakke, die Modder- en Rietriviere, in die invoer tot die modelle in te sluit. Twee opsies is ondersoek, om dit as multi-dimensionele invoer vir 'n konvolusionele netwerk in te sluit, en die gebruik van 'n parralelle argitektuur. Die opsies het onderskeidelik absoluut gemiddelde persentasie-foute van 21.6 % en 23.5 % gelewer. Alhoewel hierdie resultate nie besonder akkuraat is nie, het die modelle wel bydrae van die Vaalrivier af relatief goed voorspel, en sal hulle prakties implimenteerbaar wees indien gepaste veiligheidsfaktore op die resultate toegepas word.

Ons het gepoog om seisoenale invloed op stroomvloei te voorspel deur addisionele data as deel van die invoer te verskaf. Deur die tyd van die jaar en die verdampingmetings van 'n nabygeleë weerstasie as invoer in te sluit het die absoluut gemiddelde persentasie-foute tot 14.4 % en 14.8 %, onderskeidelik verminder (vanaf 18.4 % vir 'n model met geen seisoenale invoer nie). Ons het waargeneem dat die fout tussen die gemete en voorspelde vloei relatief konstant bly oor tyd en het daarom die foute wat in die onlangse verlede gemaak is, ingesluit in latere voorspellings. Hierdie addisionele invoer het die model se absoluut gemiddelde persentasie-fout verminder na 10.2 %, 'n beduidende verbetering op ander metodes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Vanderkloof Dam, situated on the Orange River, South Africa, supports approximately 80 000 ha of irrigated agricultural land as well as a number of towns along the 1 400 km river reach downstream of the dam. Where the river reaches the Atlantic Ocean, it fans out to form the delicate Orange River Mouth Estuary, a Ramsar site which is home to 14 endangered bird species and a bird population that can peak at more than 25 000 (Ramsar, n.d.). After too little water was released from the Vanderkloof Dam in 1995, the estuary's salt marshes collapsed and it was placed on the Montreux Record, highlighting its vulnerability to disruption by human development.

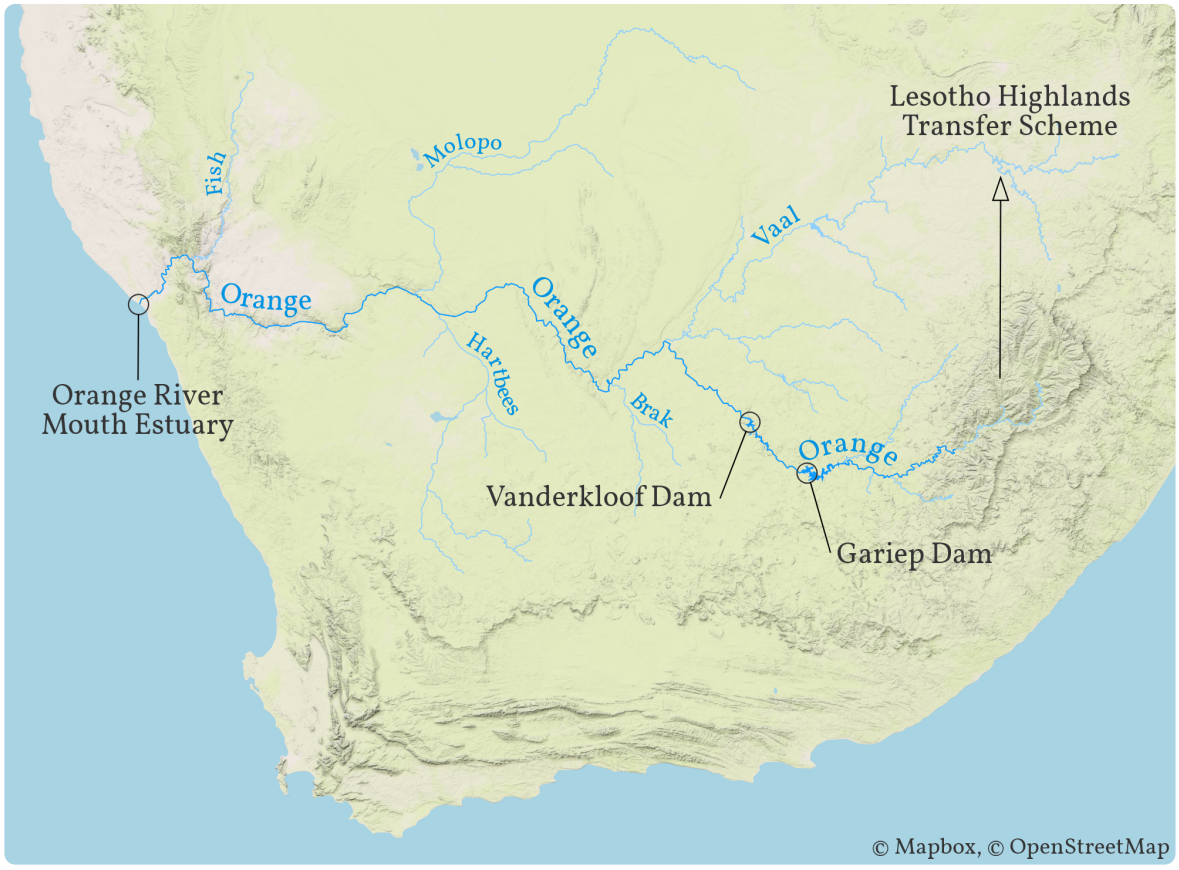


Figure 1.1: Location of the Vanderkloof Dam on the Orange River.

The location of the Vanderkloof Dam on the Orange River is indicated in Figure 1.1. No major reservoirs exist downstream of the dam and little water is contributed from tributaries between it and the river mouth. This means that the arid area surrounding the river relies solely on water released from the dam. It also means that if more water is released than is required for downstream users and losses, this fresh water is irretrievably lost into the Atlantic Ocean. A study by the South African Department of Water and Sanitation (DWS) (Maré and Sejamoholo, 2010) indicates that approximately 280 Mm^3 of water is lost because of inefficient releases from the dam each year; water which could be utilised elsewhere. This volume of water could irrigate more than 15 000 ha of agricultural land or meet the entire annual water demand of a city the size of Cape Town.

In contrast with its relatively undeveloped downstream reach, the Orange River upstream of the Vanderkloof Dam is dramatically altered from its natural state. South Africa's largest dam, the Gariep Dam, is situated approximately 130 km upstream of the Vanderkloof Dam. Further upstream at its origins in the mountains of Lesotho, the Lesotho Highlands Water Project (LHWP) transfers approximately 780 Mm^3 of water per annum to the Vaal River catchment (Lesotho Highlands Development Authority, n.d.). With phase 2 of this project due to be completed around 2025, this amount will increase to 1 270 Mm^3 per annum, effectively reducing the annual inflow into the Gariep and Vanderkloof Dams by 490 Mm^3.

It is clear that there is a need for better management of releases from the Vanderkloof Dam; if too little is released farmers would not be able to irrigate their crops and the Orange River Mouth Estuary may be permanently damaged, if too much is released significant wastage of water resources will occur. To strike this balance, and inform the DWS on the optimal amount of water to be released, a robust and accurate flow routing model that predicts the downstream effects of releases from the Vanderkloof Dam, is required.

1.1 Flow routing

Flow routing is a method that predicts the flow at a downstream point in a river, given a set of conditions upstream of that point. Figure 1.2 indicates the factors that influence routing along a river reach.

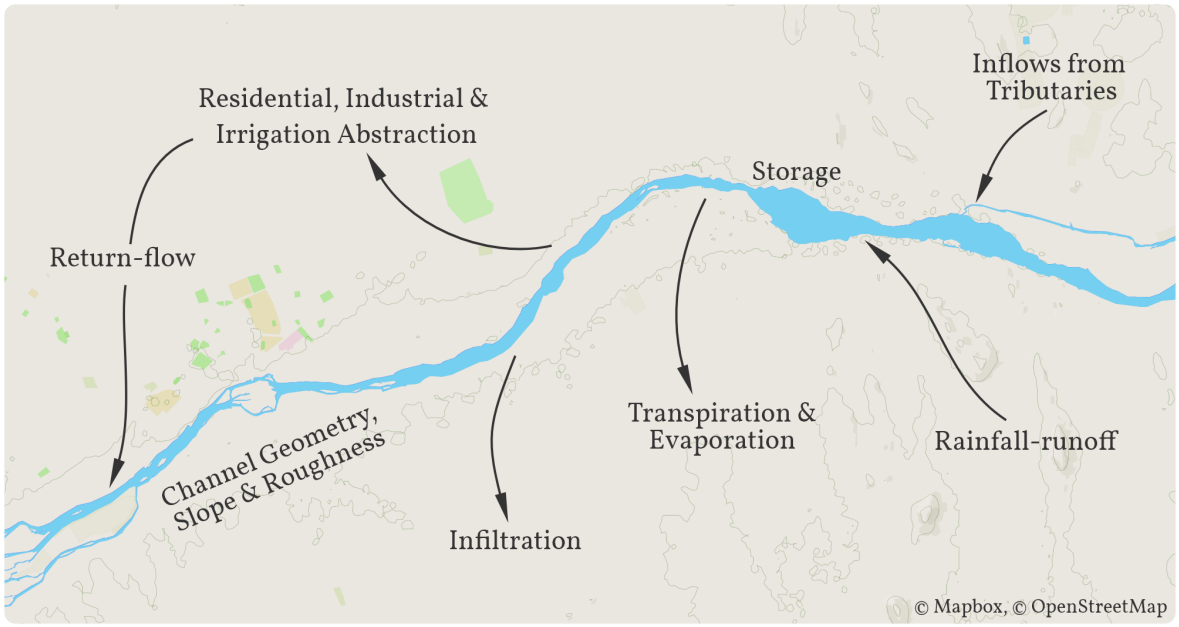


Figure 1.2: Factors that influence flow routing.

In a closed system, where detailed data is available for all influencing factors, flow routing is a relatively simple problem to solve with theoretical models that can quantify the effect for each of these factors. When detailed data is lacking, a short river reach up to a few kilometres can be approximated as a closed system and the effects of all factors other than the channel geometry, slope and roughness can be considered negligible. On a longer river reach, however, the residential, industrial and irrigation abstractions, evaporation, transpiration, infiltration, rainfall-runoff and return-flow factors make a significant difference to the downstream flow and cannot be disregarded. These additional factors are highly variable both spatially and temporally, and therefore a theoretical solution is often intractable. Chapter 2 details the underlying physical drivers and the effects that each of these factors have on the Orange River, and explains why a theoretical solution would be impractical in this particular case.

1.2 Operation of the Vanderkloof Dam

Currently the DWS sets up a minimum release schedule for the Vanderkloof Dam on an annual basis. As an example, the minimum release schedule for 2008 is displayed in Figure 1.3. The schedule shows how the water requirements are drastically higher during summer months, peaking at 110 m^3/s in December, and its lowest requirement of 28 m^3/s in June. This high variability is not only driven by higher irrigation requirements in summer, but also by the significant evaporation and transpiration losses from the water surface and riparian vegetation during those times.

Stellenbosch University https://scholar.sun.ac.za

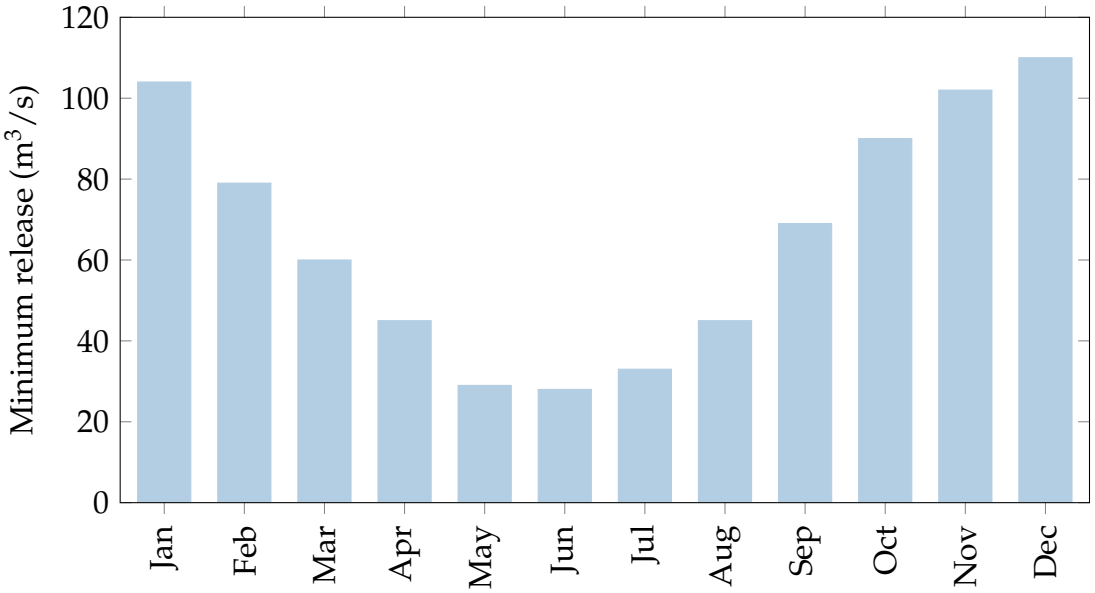


Figure 1.3: Vanderkloof Dam release schedule. Data is for the year 2008, from Maré and Sejamoholo (2010).

Water is released from two hydro-electric turbines at the Vanderkloof Dam, each having a capacity of approximately 200 m^3/s; a maximum of 400 m^3/s can therefore be released from the turbines. As can be seen from the release schedule, this is far above the rate of water required by downstream users and therefore the turbines are only opened for a set number of hours each day. Even though the DWS sets the amount of water to be released, it is Eskom's Peaking Division that manages the operation of the turbines. The current operational rules allow Eskom to decide when to open the turbines, which would typically be during periods of peak electricity demand in the mornings and late afternoons. Eskom is also allowed an annual discretionary volume of water that can be released over and above the amount defined by the schedule. This is typically used for emergency power generation.

1.3 Existing Orange River routing models

Two predictive flow routing models have previously been developed for the Orange River downstream of the Vanderkloof Dam: the first was as part of a study by the Water Research Commission (WRC) (Fair, 2003), and the second model was commissioned by the DWS (Le Grange et al., 2009). Both these models followed a similar approach where all factors affecting routing along the river were estimated and used as input for a one-dimensional hydro-dynamic model. This model in turn predicts how releases from the Vanderkloof Dam would propagate downstream. Both these models rely on assumptions about the highly variable abstractions and losses along the river. Because of this, and the extensive engineering knowledge required to operate such systems, neither of these studies produced a model that could be used as an operational tool by the DWS.

1.4 Machine learning

The term "machine learning" was coined by Arthur Samuel (1959) in a paper that described how a computer program was developed that taught itself to play the game of checkers. Within several hours of playing against itself, this program reached a level that surpassed Samuel's own abilities. Many of the recent advances in machine learning come from deep learning, a branch of machine learning that uses deep artificial neural networks (ANNs). The ANN architecture has been used in practical applications since 1989 when Yann LeCun developed a model to read handwritten digits on bank cheques (LeCun et al., 1989). Since then deep learning techniques surpassed human-level performance in games such as chess and Go (Silver et al., 2017), and are approaching human-level performance at object recognition in photographs (He et al., 2015b) and voice recognition (Xiong et al., 2018).

A short history of deep learning

The first incarnation of an ANN was called a "perceptron" (Rosenblatt, 1958). Perceptrons are loosely based on how neurons function in the brain; with the dendrites of the neuron representing the inputs and the axons representing the output as displayed in Figure 1.4.

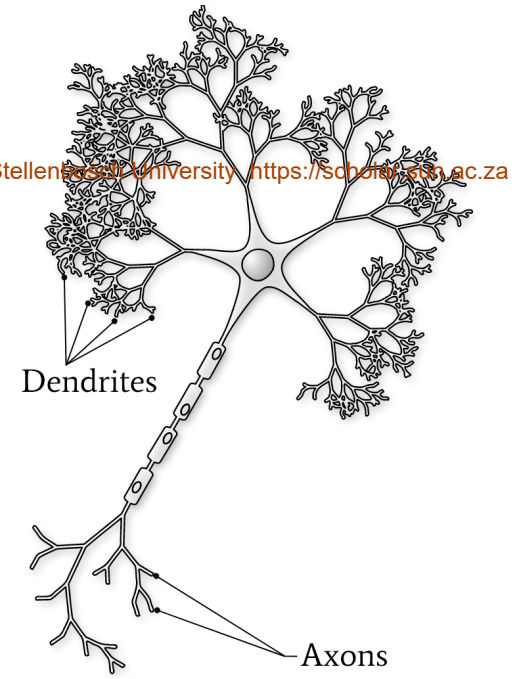


Figure 1.4: Biological neuron (by Nicolas Rougier in the public domain).

A perceptron, displayed in Figure 1.5, is activated when the weighted sum of its inputs is greater than zero. This activation is analogous to a neuron receiving inputs from its dendrites, firing and thereby transmitting a message to other neurons via its axons. For a perceptron the function that determines if it "fires" is defined by:

$$f(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{1.1}$$

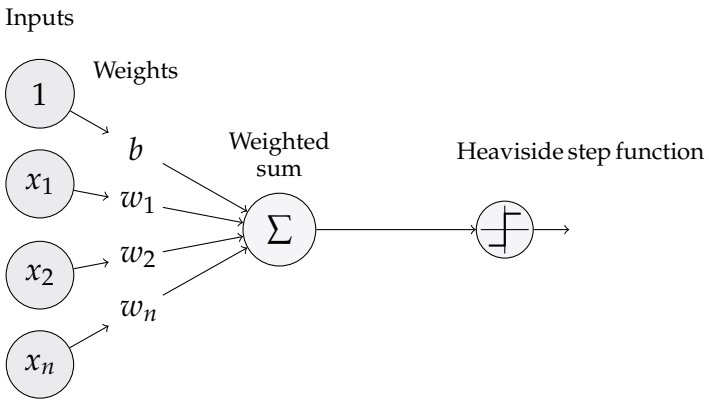


Figure 1.5: A perceptron.

Perceptrons can learn the weights (the **w** vector) and the bias (b) from observed data and then make predictions for any set of inputs (the **x** vector). The ancestors of ANNs only had the ability to learn linear dependencies between inputs and outputs (or linearly separable patterns) and they could not, for instance, learn how to predict the output of a simple exclusive disjunction[1] function (Minsky and Papert, 1969).

The limitations of single-layer perceptrons were overcome when multi-layer perceptrons (or feedforward neural networks) were developed. These networks consist of multiple layers of connected perceptrons that feed from one layer into the next. Neural networks also use non-linear activation functions, rather than the stepwise linear Heaviside step function used by a perceptron. Multiple layers and non-linear activation functions allow these networks to be applied to patterns that are not linearly separable.

Unfortunately, training these multi-layer models proved to be increasingly difficult as more layers were added. In 1986 Geoff Hinton, along with co-authors David Rumelhart and Ronald Williams, published a paper (Rumelhart et al., 1986) showing how an algorithm that back-propagates the errors through the network could be used to train an ANN with many layers in an iterative way. This became known as the backpropagation algorithm and marked the advent of deep learning. The exponential increase in computer hardware performance since the year 2000, specifically graphical processing units (GPUs), allowed deeper and deeper networks to be trained which could model increasingly complex systems.

Applying machine learning to river flow routing

The application of ANNs to the field of hydrology is not a new development. Predictive models for rainfall-runoff, flow routing, water quality as well as groundwater quality modelling have been researched since the early 1990s (American Society of Civil Engineers, 2000a; American Society of Civil Engineers, 2000b). With the diligent collection of hydrological data by the DWS, the data required to train neural networks have been steadily increasing. Along with the improvements in deep learning and the growth in computing power, the availability of suitable training data has also been a major driver in the success of deep learning. This is because the accuracy of models is often directly related to the quantity and quality of training data that is available.

[1] $f(a,b) = \begin{cases} 1, & \text{if } a \neq b \\ 0, & \text{if } a = b \end{cases} \quad a,b \in \{0,1\}$

1.5 Aim of this study

The ability to model the effects of releases from the Vanderkloof Dam could save millions of litres of water annually and free up additional water resources for South Africa. This study aims to develop a practical and robust model that can be used in the day-to-day operation of the Vanderkloof Dam and facilitates the optimisation of releases.

In the case of the Orange River, it is hypothesised that the application of machine learning, specifically multi-layer ANNs, to develop a flow routing model is particularly suited to the problem due to the complexities and the high number of unknowns in abstractions and losses along the river reach. When using an ANN model we would essentially let the data inform the model and make no explicit assumptions about the magnitude of these influencing factors. The study will aim to use only upstream flow measurements as input to predict downstream flow and include no data that is not readily available in a day-to-day operational setting.

It is hypothesised that an ANN would:

- "learn" how the underlying physical attributes affect the flow along the Orange River;
- be simpler to implement than a hydraulic model;
- produce more accurate results than a hydraulic model of the same river reach.

As part of the study we will assess the performance of a number of deep learning architectures, such as standard fully-connected neural networks, CNNs and RNNs in order to identify the most appropriate approach. We also aim to develop a pragmatic methodology to account for seasonal losses and abstractions and inflows from tributaries to make the model an end-to-end decision support tool.

Chapter 2

Factors influencing flow routing

An analysis of the underlying factors that influence flow routing along the Orange River can inform decisions about how a predictive model is set up. This chapter explores these drivers and identifies appropriate inputs into the model. For the purpose of discussion, the factors previously illustrated in Figure 1.2 are grouped into three types:

- Factors that influence the flow rate: channel geometry, slope, roughness, surface water storage and groundwater-surface water interaction.
- Factors that increase the flow volume: rainfall-runoff and inflows from tributaries.
- Factors that decrease the flow volume: evaporation, transpiration, residential, industrial and irrigation abstractions.

2.1 Factors that influence flow rate

Channel geometry, slope and roughness

The rate at which water propagates along an open channel, such as a river, is primarily determined by the channel's geometry, its slope and the roughness of the river bed. For engineering applications the relationship between the flow rate and these factors can be estimated by the one-dimensional Saint-Venant equations (De Saint-Venant, 1871), which specifically apply to one-dimensional incompressible shallow water flow in open channels:

$$\frac{\partial A}{\partial t} + \frac{\partial (Au)}{\partial x} = 0, \tag{2.1}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + g\frac{\partial \zeta}{\partial x} = -\frac{P}{A}\frac{\tau}{\rho}. \tag{2.2}$$

The variables in Equations 2.1 and 2.2 are defined as follows:

- Location (x): Distance along river reach.
- Time (t): Time at which variables are evaluated.
- Water cross-sectional area (A): Cross-sectional area of water at a particular cross-section.
- Water surface elevation (ζ): Distance between the lowest part of the river bed and the water surface.
- Wetted perimeter (P): Distance along a cross-section of the river bed "wetted" by water.
- Flow velocity (u): Speed that water flows along the length of the river.
- Shear stress (τ): The frictional force that the river bed has on the water moving over it.
- Fluid density (ρ): Approximately $998\,\mathrm{kg/m^3}$ for water in typical river flow conditions.
- Gravitational acceleration (g): Approximately $9.81\,\mathrm{m/s^2}$ at sea level.

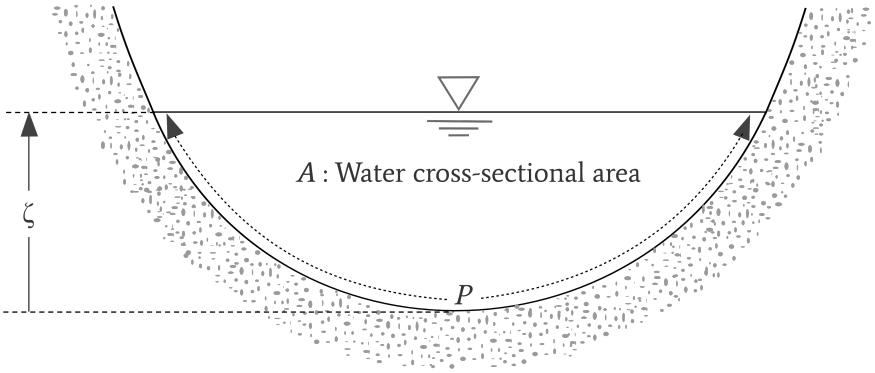


Figure 2.1: Illustration of variables in the Saint-Venant equations: water surface elevation (ζ), wetted-perimeter (P), water cross-sectional area (A).

Practically, in order to solve the Saint-Venant equations, the following data is required:

- Channel geometry consisting of cross-sections of the river channel at regular intervals which provide the values for water surface elevation (ζ), wetted perimeter (P) and cross-sectional area (A).

- Roughness coefficients for the river bed which supply the value for shear stress (τ).

Channel geometry

Figure 2.2 illustrates the dramatic variation of the geometry of the river channel along the Orange River's length.

Aerial photographs by Google, AfriGIS (Pty) Ltd, CNES/Airbus, DigitalGlobe, Landsat/Copernicus.

Figure 2.2: Variation in channel geometry: aerial views of locations just downstream of Vanderkloof Dam, the Kakamas Region and the Northern Cape respectively.

Obtaining the channel geometry typically involves employing land surveyors to conduct cross-sectional surveys at multiple locations. The most detailed model of the Orange River to date includes 124 surveyed cross-sections (Le Grange et al., 2009). Of these cross-sections, 81 are concentrated along only 86 km of the river, leaving 43 cross-sections to cover the remainder of the 1 400 km of the river reach (an average of approximately one cross-section every 30 km). Difficult access to the river at many locations, as well as the high cost, makes surveying a large number of cross-sections impractical.

Roughness coefficients

The roughness coefficient of a river bed varies not only along the length of the river, but also from one side of the river to the other. The main river channel typically has a lower roughness coefficient than the vegetation-covered river banks. When setting up a hydraulic model, roughness coefficients are typically estimated for each cross-section as part of the calibration of the model and adjusted until the errors between modelled hydrographs and the recorded hydrographs at flow gauging stations along the river reach are minimised.

It is hypothesised that a machine learning model could learn to estimate the combined effect of channel geometry, slope and roughness based on historic flow measurements.

Surface water storage

The Orange River does not have any large reservoirs downstream of the Vanderkloof Dam. The largest dam, the Boegoeberg Dam, has a storage capacity of only 20 Mm^3 and effectively functions as a large weir (Department of Water and Sanitation, n.d.[a]). Because of the limited storage volume along the Orange River, the effect of surface water storage can be ignored when setting up a flow routing model of the Orange River.

Groundwater-surface water interaction

Groundwater-surface water interaction determines the net effect of infiltration of water from the river into the river bed and return-flow (or exfiltration) from the river bed into the river. According to Kalbus et al. (2006) the magnitude of this interaction depends on the hydraulic head, which in turn is determined by the water table and the geomorphological features of the river bed. The water table at a specific location is influenced by rainfall events as well as irrigation in the surrounding area. Its effects are illustrated in Figure 2.3.

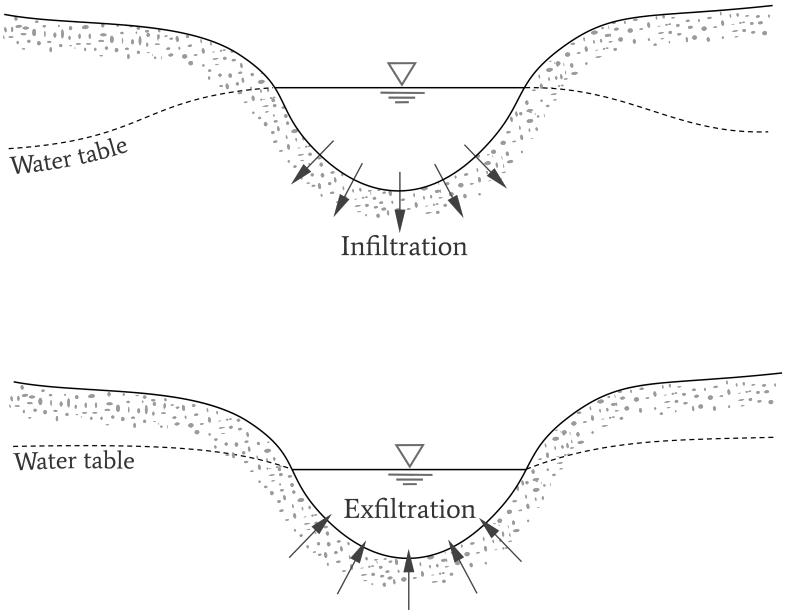


Figure 2.3: Groundwater-surface water interaction.

Return flows from irrigation are significant, and are estimated to be between 10 % and 20 % of the irrigation abstractions (Le Grange et al., 2009). In order to predict the volume of water that returns to the river and the delay between when crops are irrigated and when the return flows reach the river, one would need to take into account

the current level of the water table as well as local geomorphological data. Producing an accurate estimate for this interaction for all irrigated areas along the 1 400 km river reach is intractable. No data related to groundwater-surface water interaction will be included in the models set up in this thesis.

2.2 Factors that increase flow

Rainfall-runoff

The mean annual precipitation over the Orange River's catchment area falls from 400 mm in the area of Vanderkloof Dam to less than 50 mm at the river mouth, as illustrated in Figure 2.4. The majority of the catchment area downstream of the Vanderkloof Dam has an annual rainfall of less than 300 mm. The timing and volume of water contributed by rainfall-runoff after a rainfall event are determined by the extent and duration of the rainfall event as well as the conditions in the catchment area. However, since rainfall downstream of the Vanderkloof Dam is low and sporadic, rainfall-runoff contributes very little to the stream flow, and will not explicitly be included as input into the models set up in this thesis.

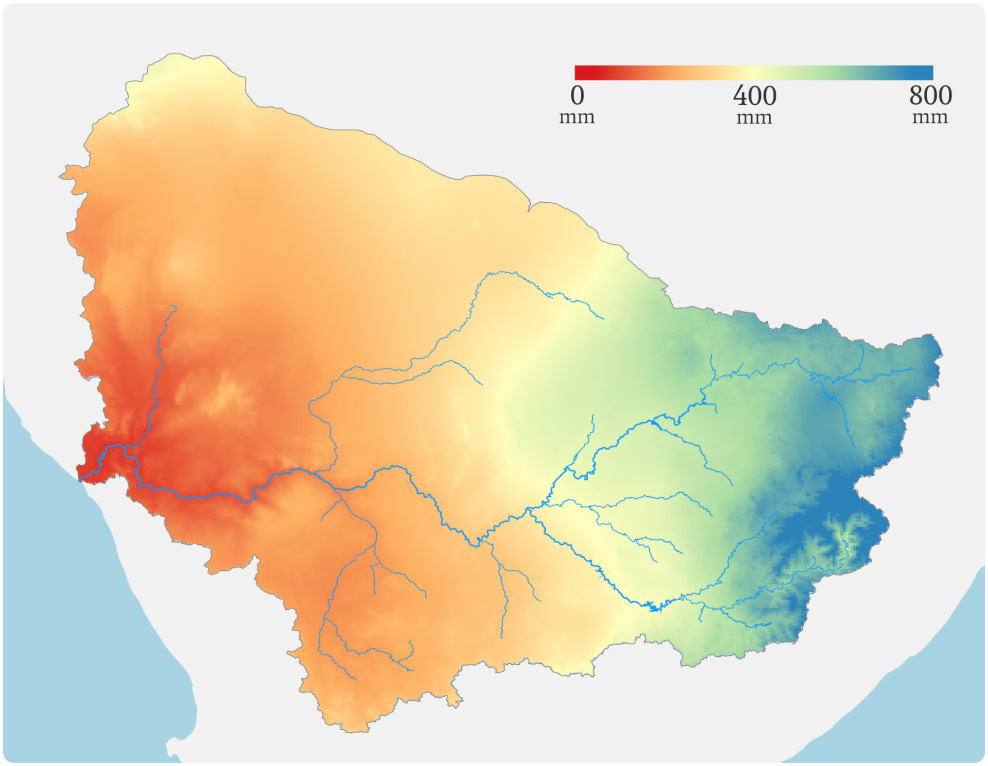


Figure 2.4: Mean annual precipitation over Orange River catchment area. Data from Fick and Hijmans (2017).

Inflows from tributaries

Contributions by tributaries downstream of the Vanderkloof Dam are determined by rainfall-runoff within each of the tributaries' catchment areas. As indicated in the previous section, the rainfall in the catchments of all tributaries other than the Vaal River is extremely low, and is therefore not considered as input into the models set up in this thesis. These contributions would, however, still fulfil the important environmental requirement of high flood-magnitude flows that seldom occur on highly regulated rivers such as the Orange River. The Vaal River is the only tributary of the lower Orange River that regularly makes a significant contribution to its flow. A simplified frequency histogram of inflows from the Vaal River into the Orange River is displayed in Figure 2.5.

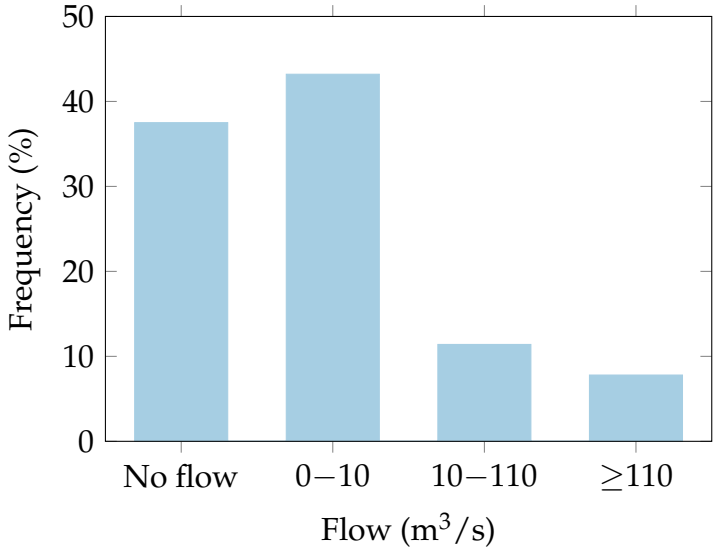


Figure 2.5: Histogram of average daily contributions from the Vaal River. Graph was generated from measurements by the DWS at Douglas Weir, 1977 to 2018.

The distribution of flow in Figure 2.5 indicates that although inflow from the Vaal River is negligibly small (below 10 m^3/s) for approximately 80 % of the time, it can contribute more than 10 % of downstream requirements the remainder of the time. It can also satisfy all downstream requirements for a small proportion of the time (approximately 8 %) when comparing these flows to the requirements from Figure 1.3.

2.3 Factors that decrease flow

Estimated requirements and losses from the Orange River used to set up the annual release schedule for the Vanderkloof Dam are displayed in Figure 2.6.

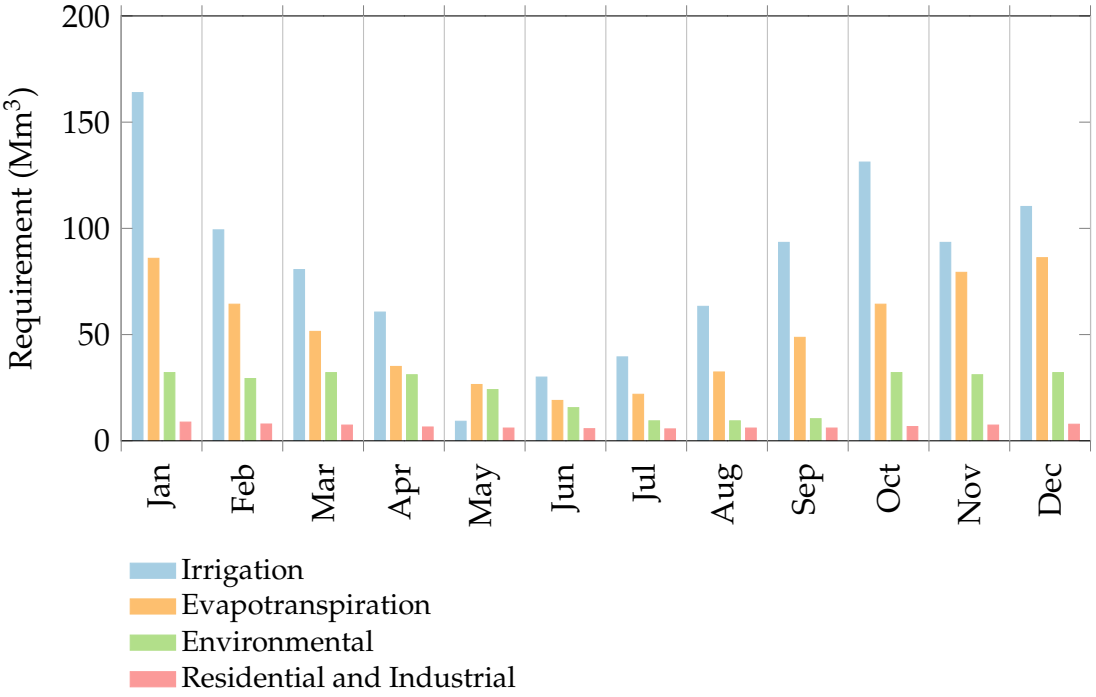


Figure 2.6: Estimated water requirements along the Orange River. Data from Le Grange et al. (2009).

Evapotranspiration

Evapotranspiration is defined as the combined effect of evaporation and transpiration, i.e. water that is lost by evaporation from the water surface of the river and by transpiration from riparian vegetation along the banks of the river. As with rainfall, these losses vary both spatially and temporally. The underlying drivers for evapotranspiration are temperature, humidity, wind speed, open water surface area and the vegetation type and extent along the river banks.

The spatial and temporal variations caused by local weather conditions are apparent when considering the recorded evaporation at three meteorological stations along the Orange River reach, displayed in Figure 2.7.

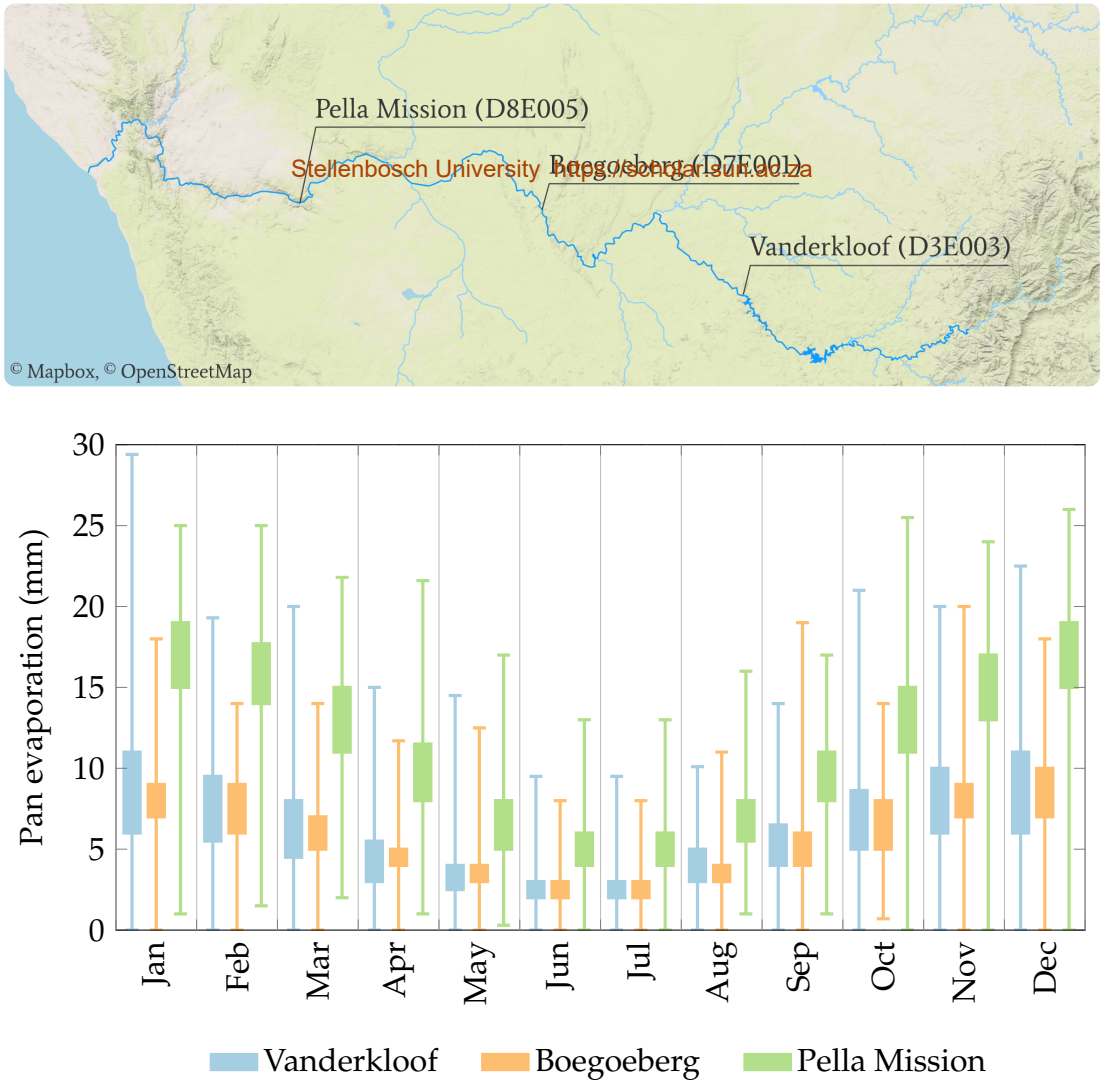


Figure 2.7: Variability of evaporation measured along the Orange River when comparing three DWS meteorological stations located at Pella Mission Station, Boegoeberg Dam and Vanderkloof Dam respectively. Differences in monthly distribution between these locations are indicated by the box-and-whisker plots. Codes in brackets are DWS reference numbers for the stations.

Weather predictions from the South African Weather Service (SAWS), which include predictions for temperature and wind speed (two major drivers of evaporation), could be included as inputs for an ANN. However, to preserve the simplicity of the models it was decided that weather predictions would only be included as input for the models if the accuracy of the simpler models was deemed too low. It is hypothesised that a machine learning model could learn to estimate the effects of evaporation loss based on recent flow measurements in the river.

Residential and industrial abstraction

Typically residential and industrial abstractions are well monitored and can be estimated with a high level of confidence. Although these values could be included in a machine learning model as inputs, they will be excluded from the models set up in this thesis in order to keep the models as simple as possible. It is hypothesised that a machine learning model could learn to estimate the effect of residential and industrial abstractions based on historic flow measurements.

Irrigation abstraction

Some irrigation abstractions are well monitored, but in general they are highly variable because they are impacted by crop choice, area under irrigation and local weather conditions. This makes predictions difficult and therefore these values will not be included as input to the machine learning models set up in this thesis. It is hypothesised that a machine learning model could learn to estimate the effect of irrigation abstractions based on historic flow measurements.

Environmental requirement

Environmental requirements are defined as the minimum average monthly flows that need to reach the Orange River Mouth Estuary to maintain its environmental integrity. These requirements are the target downstream flows that the optimisation would attempt to achieve while minimising releases from the Vanderkloof Dam.

2.4 Summary

After analysing the factors that influence flow routing along the Orange River, it is apparent that explicitly including each of these factors as inputs into a model would result in a very complex solution. Our approach will be to construct the simplest model which only includes recently measured flows along the river and releases from the Vanderkloof Dam, and then add additional inputs only if a sufficiently accurate predictive model could not be produced.

Chapter 3

Theory of deep learning

Artificial neural networks were inspired by the function of a biological nervous system, with each node in a layer representing a neuron, and the connections (or edges) between nodes representing the transmission of signals to other neurons. This metaphor of a neural network functioning as a brain is a striking one, but it has its limitations and its utility does not extend to explaining the mathematics that allow a neural network to function. This chapter aims to expand on the theory underlying neural networks and the effects of parameter choices that are made when designing these models.

3.1 Defining the problem

When setting up an ANN, we must first establish what type of learning is involved and the output required from the model. The type of learning can be classified as either supervised, semi-supervised or unsupervised, depending on the available training data and the problem domain.

- **Supervised learning** is the process of training an ANN with a set of known input-output samples. These samples are referred to as labelled data, since every input is "labelled" with an output. In this thesis we apply supervised learning to the flow routing problem since we have known input (recorded flow upstream in the river) and known output (recorded flow downstream).
- **Unsupervised learning** is applied when the output is not known and only a large amount of unlabelled input data is available. This type of learning finds its use in categorising or finding associations between inputs; a practical example being anomaly identification used for fraud detection.
- **Semi-supervised learning** is applied when there is a small amount of labelled data and a large amount of unlabelled data available. Researchers have found that pre-training a model on the unlabelled data and then training it on the small amount of labelled data significantly improves its accuracy, compared to only training it on the labelled data (Zhu, 2006).

After establishing the type of learning, we also need to identify the type of output required from the model. The output of the model could either be a continuous variable (such as house price values), or a category (such as a specific dog breed). Models that output a continuous variable are said to solve regression problems and those that output one of a number of categories solve categorisation problems.

Prediction of river flow would be most aptly framed as a regression problem, since river flow is a continuous variable. It would be possible to frame flow routing as a categorisation problem by dividing the recorded flows into buckets (or ranges) of flows, but preliminary investigations into this did not yield any improvements in performance. For this reason, the models set up to predict flow in this thesis all solve for a continuous variable.

Model architecture

There are a large number of neural network architectures, each with their own strengths and weaknesses. In each of the following sections we will investigate three neural network architectures and the mathematics that underpin them: fully-connected networks, convolutional neural networks (CNN) and recurrent neural networks (RNN). We will also discuss the choice of hyper-parameters, i.e. additional model parameters, such as the type of activation function and number of hidden layers, that need to be considered when setting up a neural network.

3.2 Fully-connected networks

A fully-connected neural network is arguably the simplest ANN architecture, with each node in a preceding layer connected to all nodes in a subsequent layer. This high level of connectivity makes this type of network very expressive, with each connection expressing a relationship between two nodes in the network. For example, there are a total of 16 connections between the first and second layer of the network displayed in Figure 3.1, and another five connections between the second and the third layers. Each connection captures the effect a node in the preceding layer has on a node in the following layer.

The variables in Figure 3.1 represent the following:

- $a_i^{(k)}$ - activation (or output) of node i of layer k.
- x_i - value of node i of the input layer. The values of the input vector can also be expressed as $a_i^{(0)}$ since the input layer can be seen as the zeroth layer in the

network.

- $w_{ij}^{(k)}$ - the weight of the connection between node i of layer $(k-1)$ and node j of layer k. Training a neural network involves changing these weights.

- nodes with 1's - bias terms that allow the network to "shift" the output of a neuron (refer to Equation 1.1 and Equation 3.1).
- $b_i^{(k)}$ - the weight of the bias value feeding into node i of layer k.

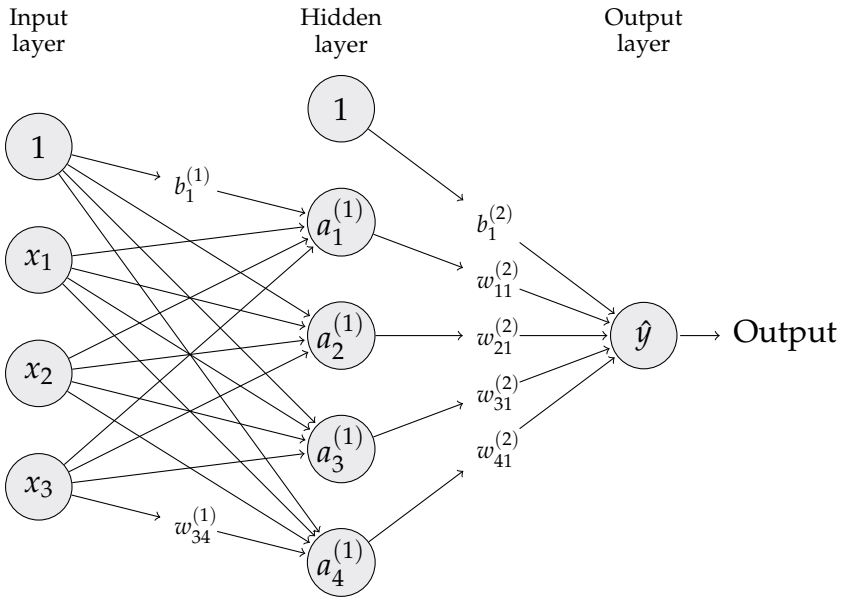


Figure 3.1: Three-layer fully-connected artificial neural network. Inputs are x_i, weights $w_{ij}^{(k)}$, bias terms $b_i^{(k)}$ and output $\hat{y}$ (see text for full description).

Predicting output with a fully-connected network

Producing an output value with a neural network is also referred to as inference and is achieved by forward-propagation of input values through the network. For a fully-connected network this involves feeding a vector representing input data into the input layer of the network and applying the following steps at each node for each layer until the output layer is reached.

1. For each node, calculate the sum $z_j^{(k)}$ of the product between the weight and the activation of the connections from the previous layer, and add the bias term (c is the number of connections to the node):

$$z_j^{(k)} = \sum_{i=1}^{c} w_{ij}^{(k)} a_i^{(k-1)} + b_j^{(k-1)}. \tag{3.1}$$

2. For each node apply an activation function g:

$$a_j^{(k)} = g(z_j^{(k)}). \tag{3.2}$$

These steps can be expressed as matrix operations.

1. Multiply the weight matrix $\mathbf{W}^{(1)}$ by the input vector $\mathbf{x}^{(0)}$ from the input layer to get the vector $\mathbf{z}^{(1)}$:

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x}^{(0)} + \mathbf{b}^{(1)}, \tag{3.3}$$

$$\begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \\ z_4^{(1)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(1)}x_1 + w_{21}^{(1)}x_2 + w_{31}^{(1)}x_3 + b_1^{(1)} \\ w_{12}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{32}^{(1)}x_3 + b_2^{(1)} \\ w_{13}^{(1)}x_1 + w_{23}^{(1)}x_2 + w_{33}^{(1)}x_3 + b_3^{(1)} \\ w_{14}^{(1)}x_1 + w_{24}^{(1)}x_2 + w_{34}^{(1)}x_3 + b_4^{(1)} \end{bmatrix}. \tag{3.4}$$

2. Apply the activation function g element-wise to the resulting vector $\mathbf{z}^{(1)}$ to calculate the activation vector $\mathbf{a}^{(1)}$ of the hidden layer:

$$\mathbf{a}^{(1)} = g(\mathbf{z}^{(1)}), \tag{3.5}$$

$$\begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \\ a_4^{(1)} \end{bmatrix} = \begin{bmatrix} g\left(z_1^{(1)}\right) \\ g\left(z_2^{(1)}\right) \\ g\left(z_3^{(1)}\right) \\ g\left(z_4^{(1)}\right) \end{bmatrix}. \tag{3.6}$$

3. Progressing to the second layer we repeat these steps and get:

$$\mathbf{z}^{(2)} = \mathbf{W}^{(2)}\mathbf{x}^{(1)} + \mathbf{b}^{(2)}, \tag{3.7}$$

$$\hat{y} = \mathbf{a}^{(2)} = g(\mathbf{z}^{(2)}). \tag{3.8}$$

For a fully-connected network, when the input is a vector, the output of each hidden layer will also be a vector. The principles of forward propagation generalises to more dimensions when multi-dimensional input arrays are used. The formulation of forward-propagation as matrix operations is, however, not only a theoretical exercise to support multiple dimensions, but also allows fast parallel execution by computer hardware. Parallelisation is possible because machine learning software libraries are optimised for matrix operations and use GPUs which can handle a large number of parallel workloads.

Interpretation of forward-propagation

We can visualise a trained ANN transforming the data representation from a preceding layer into a different data representation in the next layer during forward-propagation. For example, an ANN trained to do facial recognition would typically learn to detect low-level concepts, such as a vertical or a horizontal line feature, in earlier layers and compound concepts, such as an eye or an ear, in deeper layers. In this way it builds up a vocabulary of complex concepts by identifying the occurrence of simpler concepts in preceding layers.

Training a neural network

If we make a prediction with an untrained neural network, the result would not be sensible. ANNs are trained how to map input to output by iterative adjustment of their internal state (the weight values from Figure 3.1). This iterative adjustment is achieved by considering a large number of inputs and outputs, and attempting to reduce the error with which they are mapped from one to the other.

In order to quantify the error of a specific prediction, we must define what is called a loss function (which can also be referred to as a cost or objective function). When training a network we essentially attempt to find a (local or possibly global) minimum of the loss function over the weight space for the training data we have available. Iterating over every combination of weight values to find the minimum would be an extremely time-consuming exercise. To iterate through 100 variations of each weight between two relatively small layers with 100 nodes each, will require us to evaluate approximately 1 million combinations. This number would grow exponentially as we add more nodes and layers.

A far more efficient approach is using a method such as gradient descent, which is generally used to train neural networks. Gradient descent finds the minimum of a function by iteratively "walking" down the gradient of the loss function. This is accomplished by:

1. finding the gradient of the loss function with respect to each of the weights at the current position in the weight space;
2. adjusting each weight in the opposite direction of the gradient and proportional to the magnitude of the gradient;
3. stopping the process once the loss fails to decrease significantly within a few iterations.

Figure 3.2 illustrates this process over a simplified contour map of loss values over a two-dimensional weight space (i.e. where there are only two weights). Since the magnitude of the steps are proportional to the gradient of the function, the rate of change will speed up when the gradient increases and will slow down as the gradient decreases.

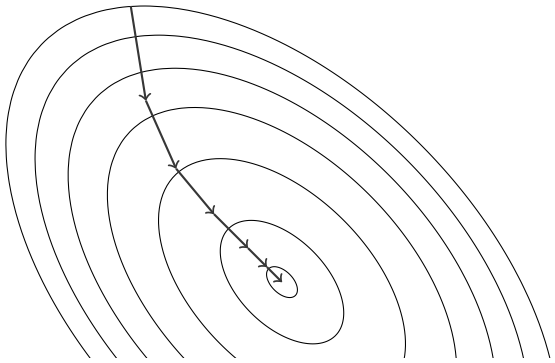


Figure 3.2: Gradient descent of loss function. The contours represent loss function values that decrease towards the ellipse in the centre of the figure. Each arrow along the path represents an iteration through the gradient descent steps.

Calculating gradients

In order to determine the gradient ($\frac{\partial J}{\partial w_i}$) of the loss function J relative to each individual weight w_i, we use the backpropagation algorithm. The application of this algorithm to neural networks was first proposed by Paul Werbos in 1974 (Werbos, 1974). To illustrate, we will apply backpropagation to a very simple two-layer neural network, displayed in Figure 3.3, with one node in the input and hidden layer each, and ignoring bias terms. All values in the diagram represent scalar values.

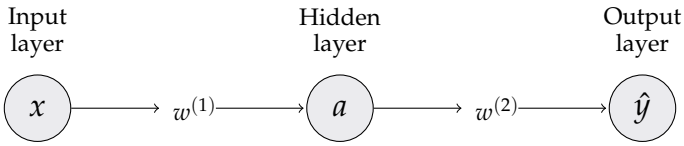


Figure 3.3: Basic single-node two-layer neural network.

Backpropagation begins with a forward-propagation step that calculates the value of the loss function. We then propagate this error from the last layer of the network back to the first layer, and calculate the gradient of the loss for each layer in turn. If we

define the loss function J for a single training sample as the mean-squared error then we have:

$$J = \frac{1}{2}(y - \hat{y})^2. \tag{3.9}$$

We multiply the error term by $\frac{1}{2}$ for convenience since it has no impact on the result of optimisation. In Equation 3.9 y is the true output and $\hat{y}$ is the output predicted by the model for a specific training sample. To calculate the partial derivative of J with respect to $w^{(2)}$ we start by applying the chain rule twice:

$$\frac{\partial J}{\partial w^{(2)}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial w^{(2)}}. \tag{3.10}$$

Combined with Equation 3.9:

$$\frac{\partial J}{\partial w^{(2)}} = -(y - \hat{y}) \frac{\partial \hat{y}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial w^{(2)}}. \tag{3.11}$$

With $\hat{y} = g\left(z^{(2)}\right)$ from Equation 3.8:

$$\frac{\partial J}{\partial w^{(2)}} = -(y - \hat{y})\, g'\left(z^{(2)}\right) \frac{\partial z^{(2)}}{\partial w^{(2)}}. \tag{3.12}$$

Since $g'\left(z^{(2)}\right)$ is the derivative of the activation function g with respect to $z^{(2)}$, the selected activation function needs to be differentiable. Choosing the identity function, $g(z) = z$, as an activation function is appropriate for the output layer when solving a regression problem since it allows us to predict a continuous variable:

$$g\left(z^{(2)}\right) = z^{(2)}. \tag{3.13}$$

Differentiating the identity function with respect to $z^{(2)}$ gives:

$$g'\left(z^{(2)}\right) = 1. \tag{3.14}$$

Since $z^{(2)} = aw^{(2)}$, the derivative of $z^{(2)}$ with respect to $w^{(2)}$ is just the activation a. Now Equation 3.12 simplifies to:

$$\frac{\partial J}{\partial w^{(2)}} = (\hat{y} - y)\, a. \tag{3.15}$$

Once we have the partial derivative of J with respect to $w^{(2)}$ we can move back and

calculate the partial derivative of J with respect to $w^{(1)}$ in the same way:

$$\frac{\partial J}{\partial w^{(1)}} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial a} \frac{\partial a}{\partial z^{(1)}} \frac{\partial z^{(1)}}{\partial w^{(1)}}. \tag{3.16}$$

Note that the first two factors in Equation 3.16 are the same as those in Equation 3.10 and do not need to be calculated again. These two terms propagated back from the previous layer are often referred to as the "error term" ($\delta^{(k)}$).

A common choice for an activation function for hidden layers is the sigmoid function:

$$g(z) = \frac{1}{1+e^{-z}}. \tag{3.17}$$

The identity and sigmoid functions are only two of many possible functions that can be selected as an activation function. This choice is discussed in more detail later in this chapter. Differentiating the sigmoid function with respect to z gives:

$$g'(z) = g(z) \cdot (1 - g(z)). \tag{3.18}$$

Therefore:

$$g'\left(z^{(1)}\right) = a(1-a). \tag{3.19}$$

Since $z^{(2)} = aw^{(2)}$, the derivative of $z^{(2)}$ with respect to a is the weight value $w^{(2)}$. Also $z^{(1)} = xw^{(1)}$, therefore the derivative of $z^{(1)}$ with respect to $w^{(1)}$ is x. Using these results and the derivative of the sigmoid function now simplifies Equation 3.16 to:

$$\frac{\partial J}{\partial w^{(1)}} = a(1-a)\, w^{(2)} (\hat{y} - y)\, x. \tag{3.20}$$

Weight update

Once we have the gradients $\frac{\partial J}{\partial w^{(2)}}$ and $\frac{\partial J}{\partial w^{(1)}}$, we proceed to the weight-update step of gradient descent. This step involves adjusting each weight w by subtracting a proportion of its corresponding gradient:

$$w^{new} = w^{old} - \eta \frac{\partial J}{\partial w}. \tag{3.21}$$

η is referred to as the learning rate and determines how fast the training process would adjust the weights. Picking a learning rate that is too high may cause gradient descent to overshoot or "bounce" around a solution, while a low learning rate may cause the model to take very long to reach it. The choice of this hyper-parameter is discussed in more detail later in this chapter.

After adjusting the weights we start another iteration of calculating the loss, back-propagating the error and adjusting the weights. The training process is typically set up so that it stops once the loss does not decrease by a certain amount for a number of iterations.

Training more complex models

In practice neural networks have multiple layers with multiple nodes in each layer, and training assesses the loss across many training samples, not just a single input-output pair. In order to account for multiple training samples, we would define the loss function (with N the number of training samples) as:

$$J = \sum_{n=1}^{N} \frac{1}{2N} (y_n - \hat{y}_n)^2 . \tag{3.22}$$

Going through the same steps for a simple model (Equations 3.12 to 3.20) it now follows that the gradient for the loss function with respect to each weight in the output layer is:

$$\frac{\partial J}{\partial w_{ij}^{(k)}} = \frac{1}{N} \sum_{n=1}^{N} (y_n - \hat{y}_n) a_{ij}^{(k-1)}, \tag{3.23}$$

and for weights in the hidden layers:

$$\frac{\partial J}{\partial w_{ij}^{(k)}} = \delta_{ij}^{(k)} a_{ij}^{(k-1)}. \tag{3.24}$$

The factor $\delta_{ij}^{(k)}$ is the error term and m the number of nodes in layer $k+1$:

$$\delta_{ij}^{(k)} = a_{ij}^{(k)} \left(1 - a_{ij}^{(k)}\right) \sum_{l=1}^{m^{k+1}} w_{jl}^{(k+1)} a_{ij}^{(k-1)}. \tag{3.25}$$

We would typically use only a portion of the training samples for a gradient descent iteration, since using the entire data set for each iteration would require a prohibitively large amount of memory and computation. The number of training samples used for an iteration is called the batch size. Once we have iterated through all training samples, one batch at a time, we have completed a training epoch. The choice of batch size and its effects are discussed in the next section about hyper-parameter tuning.

Hyper-parameters

When setting up a fully-connected network, we need to make design decisions about the following hyper-parameters:

- number of layers and nodes in each layer,
- activation functions used for each layer,
- weight initialisation,
- loss function,
- learning rate,
- training batch size, and
- gradient descent optimisation.

Since these parameters significantly influence the performance of a model they need to be chosen with care. In the next few sections we will discuss the effects of the listed parameters and how to make appropriate decisions about each of them.

Number of layers and nodes

Increasing the number of layers and number of nodes allows the network to model increasingly complex relationships between input and output. This additional expressive power comes at a cost, since training time increases as the number of learnable parameters increase. If computing power is a limitation, a model could be simplified by removing layers or nodes and thereby reducing its training time.

If the amount of training data is low, having a very complex model with a lot of expressive power may lead to overfitting - when the model starts memorising specific training data rather than generalising the underlying relationship between input and output. A model that overfits is said to have high variance.

On the other hand, if a model has too few layers (or nodes per layer), it could cause the model to underfit the data. This would cause a model to miss some important relationships between input and output. Such a model is said to have high bias. We need to adjust the number of layers and nodes in each layer and find a balance between these two extremes.

Activation functions

As discussed in Section 3.2, an activation function can be defined for each layer. These activation functions are typically non-linear, which allows the network to model non-linear relationships between input and output. Other than the sigmoid function mentioned earlier in this chapter, a common non-linear activation function is the rectified

linear unit (ReLU) function:

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0. \end{cases} \tag{3.26}$$

The ReLU function has the advantage of faster training compared to the sigmoid function, and is used extensively for networks set up in this thesis. Since the ReLU function is not differentiable at $x = 0$, a faux-derivative of zero is typically used during back-propagation.

In contrast to non-linear activation functions, linear activation functions are typically only used in the output layer, specifically for regression problems. Since this thesis focuses on predicting a continuous variable (flow), we used a linear identity function as the activation function of our final layer throughout this thesis.

Weight initialisation

Assigning a zero weight to each node in a layer would cause the weights to be adjusted uniformly during training. This would cause all weights in a layer to remain identical and severely reduce the expressiveness of the model (Fei-Fei and Karpathy, 2015). For this reason it is important to initialise weights randomly. This can be achieved using Xavier initialisation (Glorot and Bengio, 2010) - sampling from a uniform distribution scaled by the square-root of the number of nodes in the previous layer:

$$w_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]. \tag{3.27}$$

In the equation above $U[-a, a]$ is the uniform distribution over the interval $(-a, a)$, and n the number of nodes in the previous layer.

Loss function

Since our aim is to predict a continuous variable, flow at a downstream point, it is appropriate to use the mean-squared error between the predicted and actual flows as a loss function (with N the number of training samples):

$$J = \frac{1}{N} \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 . \tag{3.28}$$

Using this loss function severely penalises large errors since the loss would be proportional to the square of the difference between the actual and the predicted flows. If we want to penalise under-prediction more than over-prediction we can use the mean-

squared logarithmic error:

$$J = \frac{1}{N}\sum_{n=1}^{N}\left(\log y_n - \log \hat{y}_n\right)^2. \tag{3.29}$$

A loss function less sensitive to occasional large errors is the logarithm of the hyperbolic cosine of the error:

$$J = \frac{1}{N}\sum_{n=1}^{N}\log\left(\cosh\left(y_n - \hat{y}_n\right)\right), \tag{3.30}$$

which, for a single training sample i, approximates the mean-squared error when the error is small and the mean absolute error when the error is large:

$$J_i \approx \begin{cases} \frac{1}{2}\left(y_i - \hat{y}_i\right)^2, & \text{if error is small} \\ \mid y_i - \hat{y}_i \mid - \log 2, & \text{if error is large.} \end{cases} \tag{3.31}$$

How these loss functions perform would depend on the training data and model architecture (and possible other factors) and need to be tested with trial-and-error. This is discussed in more detail in Chapter 5.

Learning rate

The learning rate refers to the rate at which weights are adapted during the training process (refer to Equation 3.21). We can assess if the learning rate is too high or too low by plotting the loss against training epochs. If the loss "jumps" around wildly and does not seem to follow a general downward trend, decreasing the learning rate may be appropriate. Conversely, if there is very little variation between losses of different epochs and the decrease in loss is very slow, increasing the learning rate should be considered. A pragmatic approach is to start with a relatively high learning rate and decrease (or decay) it when the loss does not decrease after a set number of training epochs.

Batch size

As mentioned earlier in this chapter, each iteration of gradient descent only uses a portion of the training samples equal in size to the batch size parameter. Increasing the batch size would improve training time since it would require less iterations to complete a training epoch, but it would also increase the computer memory required during training.

Gradient descent optimisation

In practise, the gradient descent algorithm is often adapted to improve training stability or reduce training time. One such adaptation which improves training stability is stochastic gradient descent. This involves selecting a training sample at random from the training set batch for each iteration of the algorithm, rather than in the order it occurs in the original training set.

Another way to improve both the stability of the training and the speed at which training converges is to apply momentum (Rumelhart et al., 1986). Using this method weights are adjusted, not only using the error for that training iteration, but also including a portion of the weight adjustment from the previous iteration.

The Adam variant of stochastic gradient descent adapts the learning rate for each of the individual weights and also employs the second-order moment of the gradient for the weight-update. This has been shown to reduce training time (Kingma and Ba, 2014) and we make use of it when training RNNs in Chapter 5.

Hyper-parameter tuning

The parameters listed above do not function in isolation and changing one could change the effects of another. This makes hyper-parameter tuning a time-consuming and manual task where the performance of multiple models must be considered, each one with a different combination of parameter values. Although there is a growing move towards automating this tuning process, the tools and computing power required to achieve this for regression problems were not readily available at the time of writing this thesis, and manual hyper-parameter tuning was employed.

Regularisation

Overfitting causes a model to fit the training data set too well, fitting individual training samples, and not generalising well to data that is not in the training data set. To counter this behaviour one can increase the amount of training data, reduce the expressiveness of the model, cease training once overfitting is detected, or apply regularisation techniques.

We apply the dropout regularisation technique (Srivastava et al., 2014) on all models trained in this thesis. Dropout involves randomly setting a portion of a layer's weights to zero during forward-propagation at training time. Dropout is not applied at prediction-time. One can now consider the trained model to be a combination, or ensemble, of smaller models that jointly produce its output.

3.3 Convolutional neural networks

Convolutional neural networks are feedforward neural networks that include one or more convolutional layers. Convolutional layers are layers where each node in a layer is only connected to a subset of nodes in the previous layer. This subset of nodes corresponds to a receptive field to which a convolution filter is applied. During inference the receptive field moves across the layer, considers a subset of the nodes and makes a localised linear transformation. Since these transformations remain the same while they are applied across a layer, CNNs are also called space invariant neural networks. When applying a CNN to time-series data it would be appropriate to call it a time invariant neural network since convolutions would be applied across a time window and not across a spatial area. Figure 3.4 illustrates how a convolution filter is applied to its receptive field in the input layer.

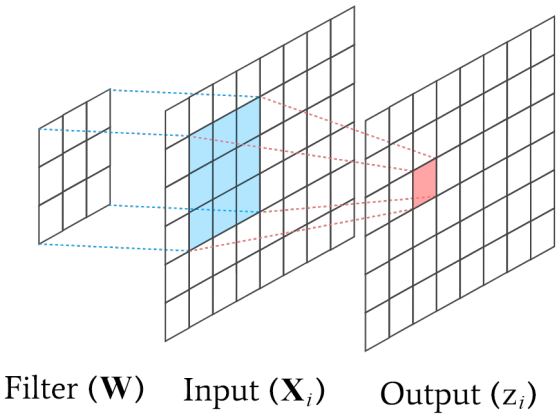


Figure 3.4: Applying a convolution filter to its receptive field (in blue) in the input layer to produce output (in red). The bias term is not displayed here.

A convolutional layer's parameters are a set of learnable convolution filters that produce localised transformations. Each of these filters would extract a certain feature from the input by producing high activation values if that feature is detected. The convolution for the receptive field i in Figure 3.4 can be expressed as a matrix operation, the Frobenius inner product, with the output z_i, filter $\mathbf{W}$, input $\mathbf{X}_i$ and bias b:

$$z_i = \langle \mathbf{W}, \mathbf{X}_i \rangle_{\mathrm{F}} + b. \tag{3.32}$$

The Frobenius inner product is the sum all elements of the element-wise (Hadamard) product of two equally sized matrices.

It is clear that there are similarities between Equation 3.32 and Equation 3.3, used by fully-connected networks, except that in this case the transformation is only applied to

the filter's receptive field to produce each output value, rather than to the entire input vector or matrix.

Interpretation of convolutions

A convolutional filter that is trained to detect a horizontal line in an image will have high activation values when its receptive field moves across a part of the image that has a horizontal line in it. Unfortunately these matrices are not often as easy to interpret as this example, especially at deeper layers, and therefore it is mostly still impossible to reverse-engineer and determine logically how a CNN produced a specific prediction.

Pooling layers

In order to reduce the output size of convolutional layers in a CNN, they are often interleaved with pooling layers. Pooling layers aggregate the outputs from the preceding layer by only propagating the maximum (or average) output of neighbouring nodes to the next layer. This aggregation effectively down-samples the output from the previous layer.

Convolutional layer hyper-parameters

When setting up a convolutional layer, we need to decide on:

- The number of output filters. This parameter also corresponds to the depth dimension of the output since there will be transformation of the input for each filter.
- The size of the filter, or receptive field (the size of the filter in Figure 3.4 is 3×3).
- The speed at which the receptive field moves across the input; this is called the stride and defines by how much a filter is shifted in between producing output values. If we set the stride to more than 1 we are down-sampling the output.
- The amount of zero padding along the edges of the input layer. Intuitively it can be seen that the output layer would always be smaller than the input layer in the width and length dimensions since a filter cannot move up to the edge of the input layer. In order to counter this reduction in resolution we can pad the edges of the input matrix with zeros to produce output with the same dimensions as the input layer.

Network architecture

A typical CNN architecture, that is still widely applied today, alternates one or more convolutional layers and a pooling layer and repeats this pattern a number of times. This architecture was initially used in the AlexNet model (Krizhevsky et al., 2012) as part of the ImageNet challenge (Russakovsky et al., 2015). The network architecture for AlexNet is displayed in Figure 3.5.

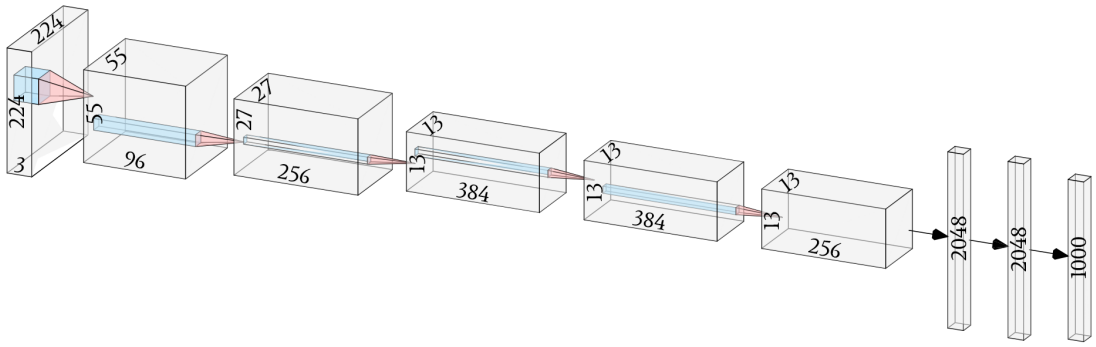


Figure 3.5: AlexNet architecture. First six layers are convolutional layers, and last three are fully-connected layers. There are pooling layers (not displayed) after the first, second and fifth convolutional layers. Figure generated with NN-SVG (Lenail, 2018).

All-convolutional network

A study by Springenberg et al. (2015) found that including a stride of two on every alternate (every second) convolutional layer instead of including the pooling layer achieves similar results. This simplifies the model and reduces training time. In this thesis we use this approach and do not make use of pooling layers.

Application to time-series data

Although CNNs are often associated with image-related machine learning problems, they can also be applied to other problem domains (Bai et al., 2018). Image input is three-dimensional, with width, height and depth (corresponding to the red, green and blue colour channels). If we set up a CNN to predict flow routing, we can simply define the input as a one-dimensional array of recorded flows at equal time intervals rather than a three-dimensional array of image data. When setting up such a model, convolutional matrices will be one-dimensional arrays that transform the one-dimensional input into a one-dimensional output for each output filter.

3.4 Recurrent neural networks

Recurrent neural networks (RNNs) are models that contain a recurrent layer - a layer that has a feedback loop and some internal state that allows it to "remember" data that was passed through it previously. This ability to remember makes RNNs especially suitable to problems with a temporal element such as speech recognition, translation, or in the case of this thesis, river flow routing. Figure 3.6 shows a diagram indicating the connection between the input, recurrent and output layers.

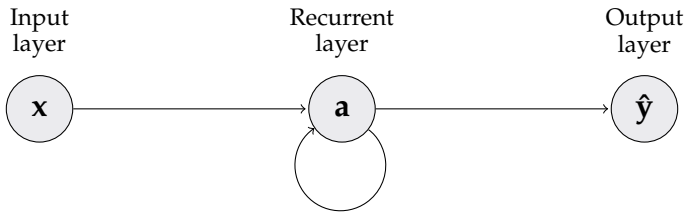


Figure 3.6: Neural network with single recurrent layer. Here the network produces an output vector $\hat{\mathbf{y}}$. The RNNs set up in this thesis will include a fully-connected layer from this output to produce scalar output $\hat{y}$.

Imagine a model that predicts the next word in a sentence given the sequence of words: "There was a storm brewing. He did not look forward to driving in the . . . ". We could imagine the likelihood of the next word being "rain" rather than the word "car" should be higher because of the context provided by the preceding sentence. In order to take into account this additional context, a network would need a memory of previous input in the sequence. The recurrent layers of a RNN have feedback connections that allow them to maintain memory of previous data that was passed through them and store this context.

Unfortunately a simple RNN with a basic tanh activation function, though theoretically capable of reacting on long-term dependencies, suffers from a deficiency called the vanishing gradient problem (Hochreiter, Bengio et al., 2001). This problem causes the training of such an RNN to stagnate because gradients are not propagated effectively back through the feedback loops, and so disappear over time. To overcome this problem a variant of the RNN architecture called long short-term memory (LSTM) was developed (Gers et al., 1999; Hochreiter and Schmidhuber, 1997). LSTMs have a gated structure, displayed in Figure 3.7, that selectively forgets, updates and uses parts of the cell's internal state to produce output.

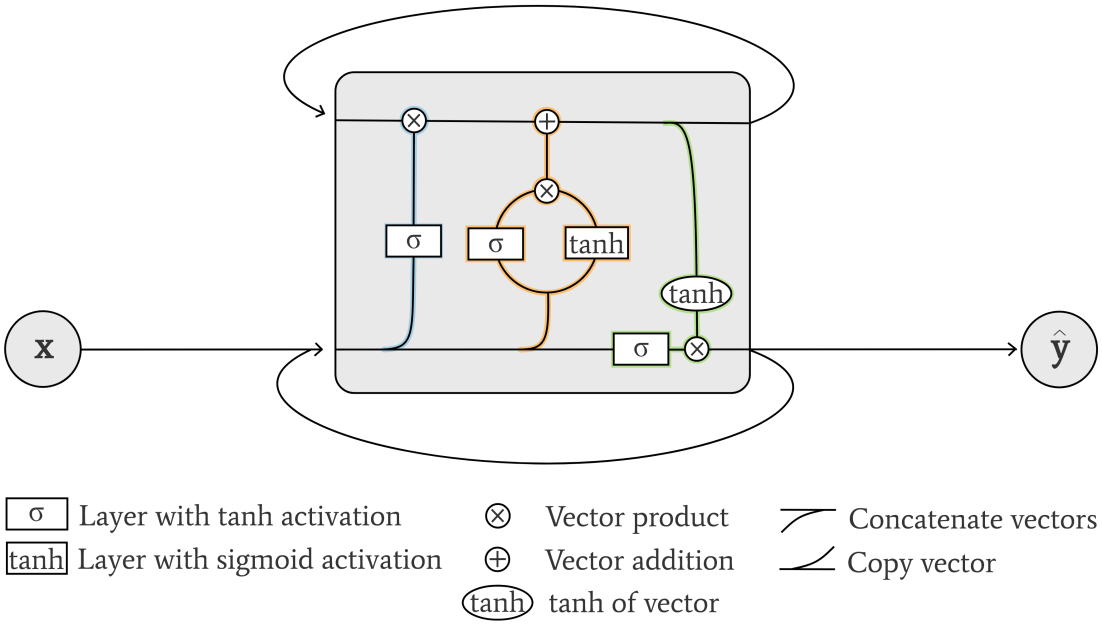


Figure 3.7: Long short-term memory cell. The forget gate is in blue, update gate in orange and output gate in green. The loop at the top represents cell state and the loop at the bottom represents the output feedback loop. Figure was adapted from Olah (2015).

The three gates of an LSTM node and their purposes are as follows.

- The forget gate (blue) decides which parts of the state to forget based on the input.
- The update gate (orange) decides how to update the cell state with new information.
- The output gate combines the cell state and the output from the input layer to produce output from the LSTM cell.

Combining these three gates inside the LSTM allows the network to overcome the vanishing gradient problem and train effectively.

A simpler variant of an LSTM, called a gated recurrent unit (GRU), combines the update and forget gates (Cho et al., 2014). Since GRUs have a simpler architecture they train faster, and have been shown to perform similarly to LSTMs in empirical experiments (Chung et al., 2014). A diagram showing the structure of a GRU is displayed in Figure 3.8. RNNs set up in this thesis use GRUs exclusively, mostly due to reduced training time that made it possible to run the models on less powerful hardware.

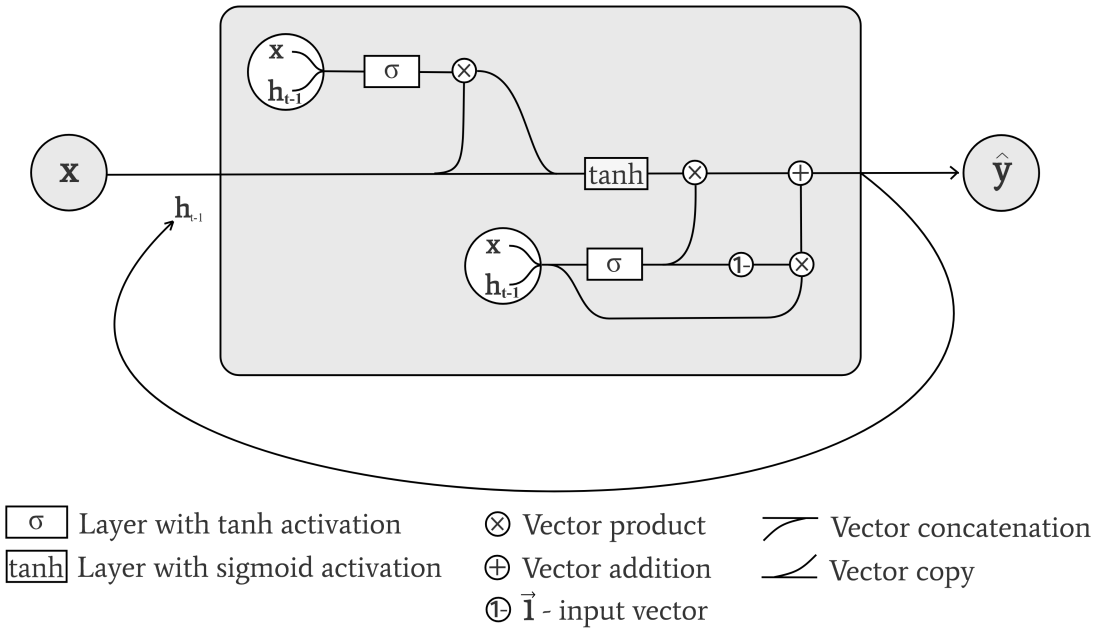


Figure 3.8: Gated recurrent unit. $\vec{1}$ represents an all-ones vector. Figure was adapted from Olah (2015).

3.5 Summary

This study will assess the suitability of applying fully-connected ANNs, CNNs and RNNs to the flow routing problem to establish the most suitable architecture and establish if a data-driven machine learning approach is a viable option for flow routing predictions along the Orange River.

Chapter 4

Data analysis and preparation

Successfully training an ANN depends largely on the quality and quantity of data used during training. The quality of the data has a direct impact on the accuracy of a model, since a model can never surpass the accuracy of the data it was trained with. Typically a model would also generalise better (when presented with non-training data) if more varied data is utilised for training. For this reason, assessing the availability, accuracy and distribution of training data is of utmost importance.

4.1 Identifying suitable sources of data

Limiting ourselves to the use of data sources that are readily available in an operational setting would ensure that the models we produce are practically implementable. Fortunately the DWS's Hydrological Services publishes a significant amount of historic and near-real-time data on their website (Department of Water and Sanitation, n.d.[b]). These data sources include the following station types:

- **Flow gauging stations** that measure the stage (water surface elevation) and the corresponding flow along rivers, canals and pipelines.
- **Reservoir stations** which measure reservoir levels and spillage over the dam walls and releases from sluices and hydro-power turbines.
- **Meteorological stations** which measure rainfall and evaporation.
- **Tidal stations** which measure estuary and lagoon levels.

The Orange River has several flow gauging, reservoir and meteorological stations which have collected data since the commissioning of the Vanderkloof Dam in 1977, and in some cases even earlier. Note that there is no tidal station at the Orange River Mouth Estuary, so this station-type was not considered further in this thesis. Data from the available stations was analysed for suitability as training data for ANNs, and the results are presented in the following sections.

Identifying suitable flow gauging stations

Over time the DWS commissioned several flow gauging stations within the Orange River catchment area. These stations were assessed spatially by plotting them on a map and colouring them based on if they are actively collecting data or not. The results are displayed in Figure 4.1.

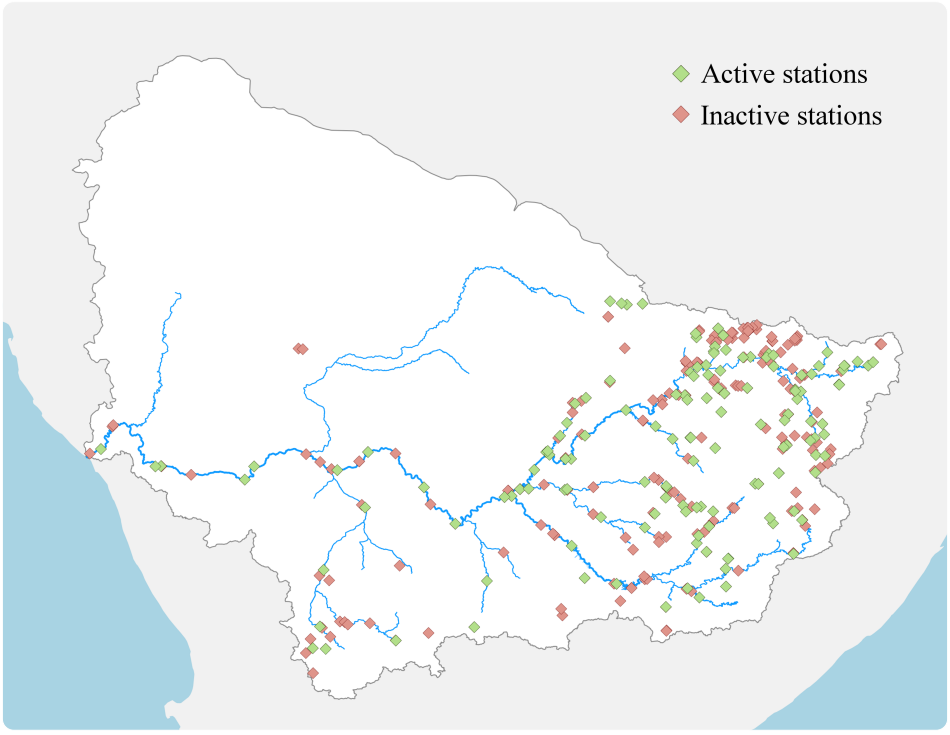


Figure 4.1: Active and inactive flow gauging stations in the Orange River catchment area.

In order to narrow down which gauging stations would be useful for providing training data, stations matching the following criteria were eliminated.

- All stations that no longer actively collect flow gauging data were eliminated. Even though a number of these inactive stations have a very long record history, including their data as part of training would not be useful since their data is not available in an operational setting.
- All stations upstream of the Vanderkloof Dam on the Orange River were eliminated. Scheduled releases from the dam control the flow downstream of the dam completely, and therefore flow upstream of the dam would have no direct effect on the lower reaches of the river.

- All stations on tributaries that join the Orange River too far downstream for contributions to be taken into account when optimising releases from Vanderkloof Dam were eliminated. This requirement eliminated all stations along tributaries other than the Vaal River.

By applying the above criteria we identified 24 flow gauging and reservoir stations at 14 locations on the Orange River that could produce useful training data. These stations are indicated on Figure 4.2.

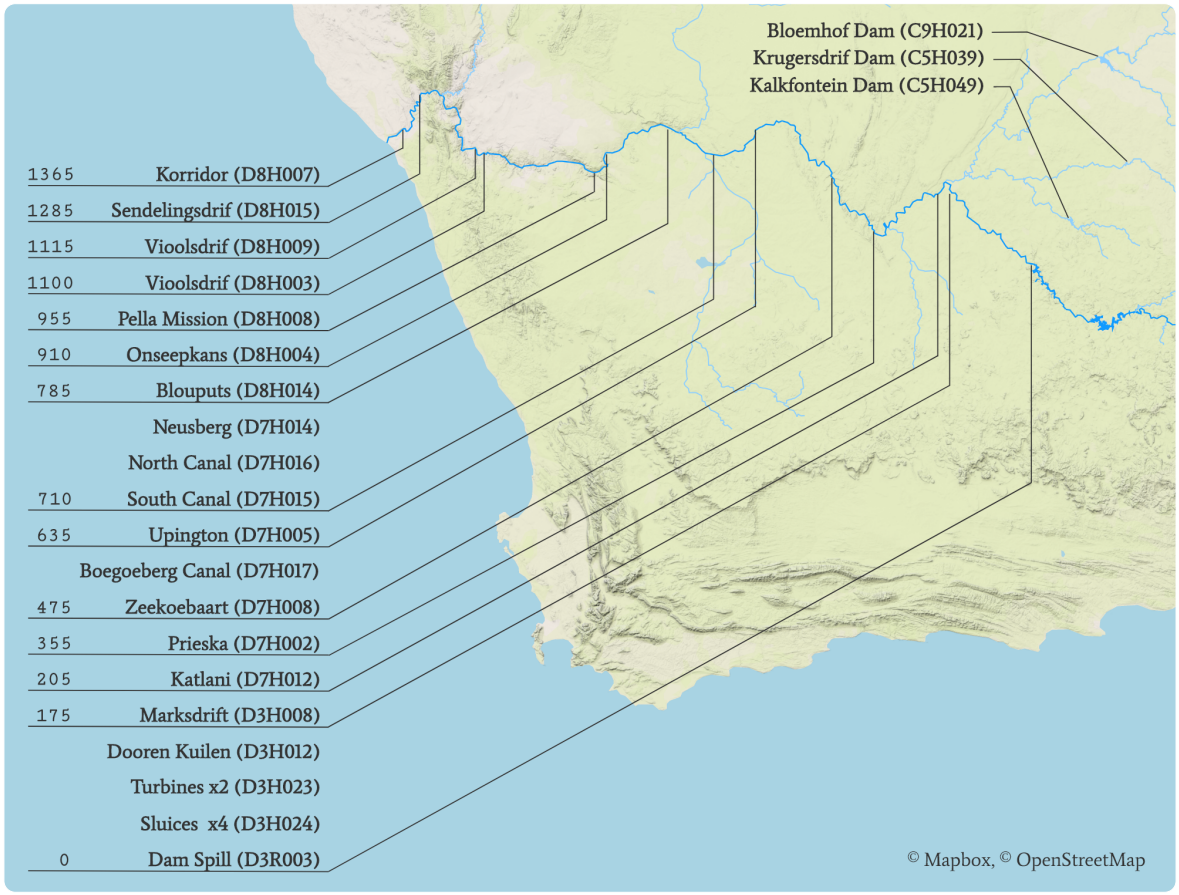


Figure 4.2: Active flow gauging stations in suitable locations along the Orange, Vaal, Modder and Riet Rivers. Codes in brackets are the DWS station numbers. Numbers on left are chainage distances along the river for each of the stations, measured from the Vanderkloof Dam.

Flow gauging stations on the Vaal, Modder and Riet Rivers

The Vaal River joins the Orange River approximately 140 km downstream of the Vanderkloof Dam, which means a model would need to pre-empt contributions from the Vaal River by predicting how much water is still approaching the Orange River. Three flow gauging stations on the Vaal River, and its tributaries the Modder and Riet Rivers, were

identified as being suitable for inclusion in the model. Their locations are displayed in Figure 4.2. These stations were chosen to be far enough upstream of the Orange-Vaal confluence so that inflows from these rivers can be taken into account when optimising releases from the Vanderkloof Dam. By including these flows one could effectively reduce the amount of water released from the dam by the amount of water that will be contributed. The hypothesis is that the model would be able to predict this contribution by considering historic measurements at these stations and the resulting contribution to the Orange River at a flow gauging station just downstream of the Orange-Vaal confluence (Katlani, D7H012).

Exclusion of recently constructed flow gauging stations

The stations at Blouputs (D8H014) and Sendelingsdrif (D8H015) are relatively new and have only collected data since 2014. Although these stations are currently actively collecting data, they might not yet have enough training data to train an ANN on. For this reason they will not be included in the training of models in this thesis, but should be considered in future once enough data has been collected.

Stations located at Vanderkloof Dam

There are a total of eight gauging stations located at Vanderkloof Dam:

- One station (D3R003) records spillage over the dam wall.
- Four stations (D3H024) record releases from the sluice gates.
- Two stations (D3H023) record releases through the two hydro-power turbines.
- Dooren Kuilen (D3H012), a station located just downstream of the dam, records the combined releases from the dam.

When setting up the ANN models we will use the measurements taken at the Dooren Kuilen station (D3H012) as the released flow from the Vanderkloof Dam, since this station effectively records the combined measurements from the other seven stations.

Identifying suitable meteorological stations

A similar approach was followed for assessing meteorological stations as the approach we followed with flow gauging stations. The locations of the meteorological stations were plotted on a map and coloured according to if they are actively recording data or not. The resulting map is displayed in Figure 4.3.

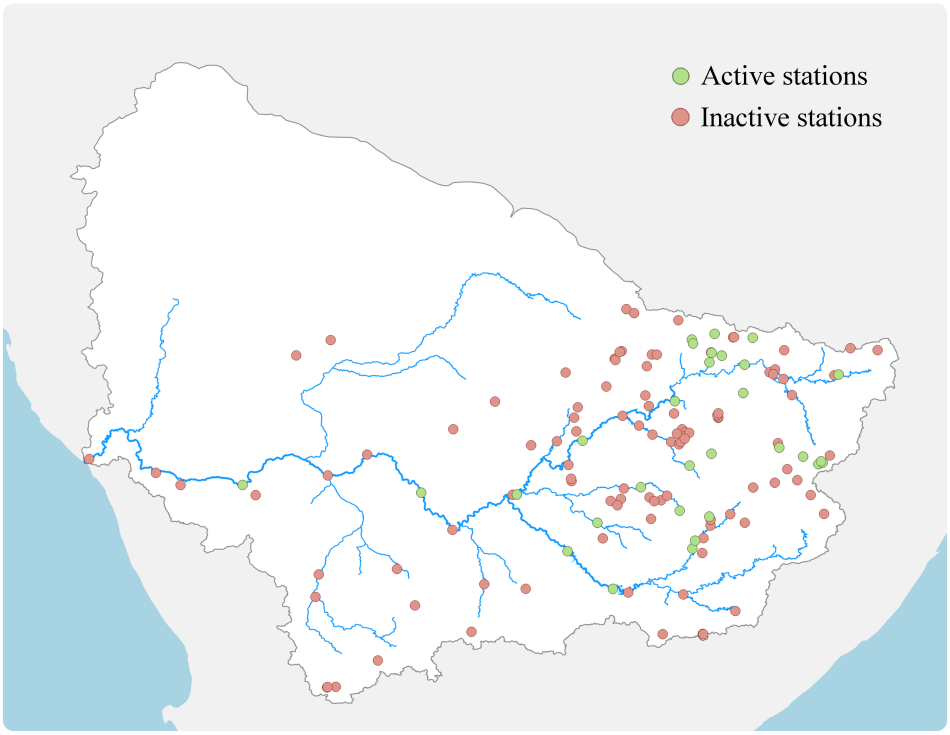


Figure 4.3: Active and inactive meteorological stations in the Orange River catchment area.

The DWS meteorological stations collect data for evaporation and for rainfall. Evaporation is measure at these stations using a Class A evaporation pan. Contributions from rainfall-runoff to the flow of the Orange River downstream of the Vanderkloof Dam are very small because of the low rainfall in the area (refer to Figure 2.4). In contrast, losses from the river from evapotranspiration are estimated to be more that 600 Mm^3 per annum for the Orange River reach downstream of Vanderkloof Dam (Le Grange et al., 2009). For this reason rainfall data was not considered further as input to our models, but evaporation data was assessed for this purpose.

Because evapotranspiration is affected by local weather conditions, only data from stations close to the river would be suitable for predicting these losses. The meteorological stations in the immediate vicinity of the Orange, Vaal, Modder and Riet River reaches are displayed in Figure 4.4.

4.2 Data quality assessment

Once the flow gauging and meteorological stations were identified that have suitable data for training our models, the raw data for these stations were downloaded from the DWS Hydrological Services website and assessed in terms of quality and distribution. Data earlier than 1977 was not downloaded since the Vanderkloof Dam's construction

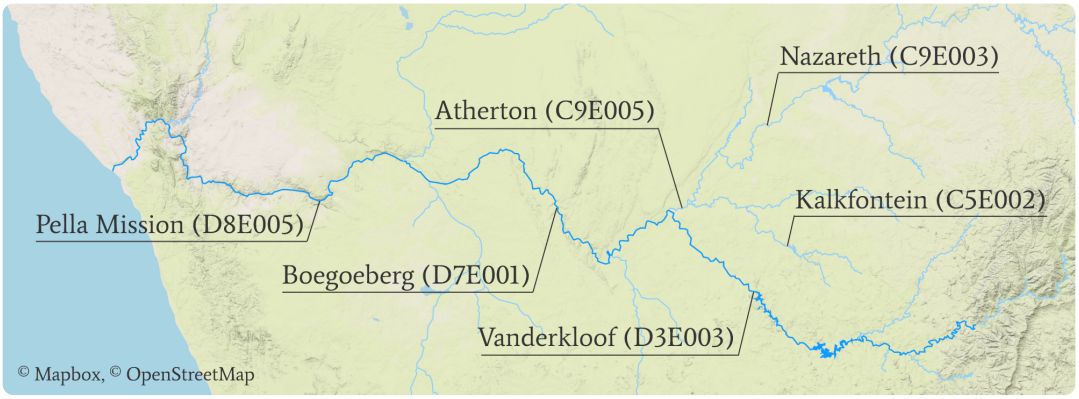


Figure 4.4: Active meteorological stations in suitable locations along the Orange, Vaal, Modder and Riet Rivers. Codes in brackets are the DWS station numbers.

was completed in 1977 and flows before this year are not considered representative of the status quo. Appendix A contains samples of the raw text data files with flow gauging, turbine, reservoir and evaporation data.

Assessment of quantity of data

All stations that contribute to a training data set must have data at intersecting times, and therefore the stations with the lowest availability of data would determine the date ranges for which training data can be produced. Even though there are significant amounts of data available at most of the flow gauging and meteorological stations, periods where no data were recorded would make it impossible to produce training data for those particular date ranges. Each of the stations' data was assessed in terms of how much data is available and also in terms of the gaps in the data set. The results of this assessment are listed in Table 4.1, Table 4.2 and Table 4.3 for flow gauging along the Orange River, flow gauging along the Vaal, Modder and Riet Rivers, and for meteorological stations respectively.

If we disregard the flow gauging station at Korridor (D8H007), the results of the data availability analysis indicate that there are more than 15 years worth of data available for all flow gauging and meteorological stations. Because stations would need data at intersecting times to produce a training sample, not having data for one of the stations means we cannot produce a training sample for that particular date range. In practice, this was not found to occur frequently, and there are significant amounts of overlapping data available in all cases.

Table 4.1: Availability of data at flow gauging stations on the Orange River. Missing data is defined as periods in the data set with no data for more than 24 hours. Pre-2005 data for D7H017 uses the code D7H013. No data was recorded at Korridor (D8H007).

Station name	Code	Start date	Total data downloaded (years)	Available data (years)	Missing data (years)
Dooren Kuilen	D3H012	1981	35.8	31.7	4.1
Marksdrift	D3H008	1977	40.6	32.8	7.8
Katlani	D7H012	1989	28.6	15.8	12.8
Prieska	D7H002	1977	40.7	32.7	8
Zeekoebaart	D7H008	1977	40.7	30.9	9.8
Boegoeberg Canal	D7H017	1977	41.7	15.1	26.6
Upington	D7H005	1977	40.7	35.4	5.3
Neusberg	D7H014	1993	24.2	20.7	3.5
Neusberg South Canal	D7H015	1993	25.1	16.8	8.3
Neusberg North Canal	D7H016	1993	25.1	16.6	8.5
Onseepkans	D8H004	1977	40.8	32.9	7.9
Pella Mission	D8H008	1979	38	31	7
Vioolsdrif	D8H003	1977	40.7	29	11.7
Vioolsdrif	D8H009	1979	38.1	31.4	6.7

Table 4.2: Availability of data at flow gauging stations on the Vaal, Modder and Riet Rivers. Missing data is defined as periods in the data set with no data for more than 24 hours.

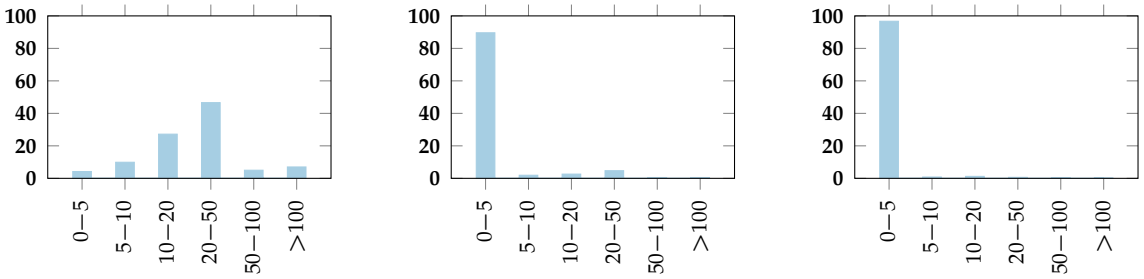
Station name	Code	Start date	Total data downloaded (years)	Available data (years)	Missing data (years)
Bloemhof Dam	C9H021	1977	41.6	21.6	20
Krugersdrift Dam	C5H039	1977	41.6	22.7	18.9
Kalkfontein Dam	C5H049	1990	28.3	21.5	6.8

Table 4.3: Availability of data at meteorological stations. Missing data is defined as periods in the data set with no data for more than 72 hours.

Station name	Code	Start date	Total data downloaded (years)	Available data (years)	Missing data (years)
Pella Mission	D8E005	1989	27.6	25.1	2.5
Boegoeberg	D7E001	1989	27.7	23.2	4.5
Vanderkloof	D3E003	1989	27.8	27.6	0.2
Atherton	C9E005	1989	27.8	23.4	4.4
Nazareth	C9E003	1970	41.6	40.5	1.1
Kalkfontein	C5E002	1970	41.5	41.2	0.3

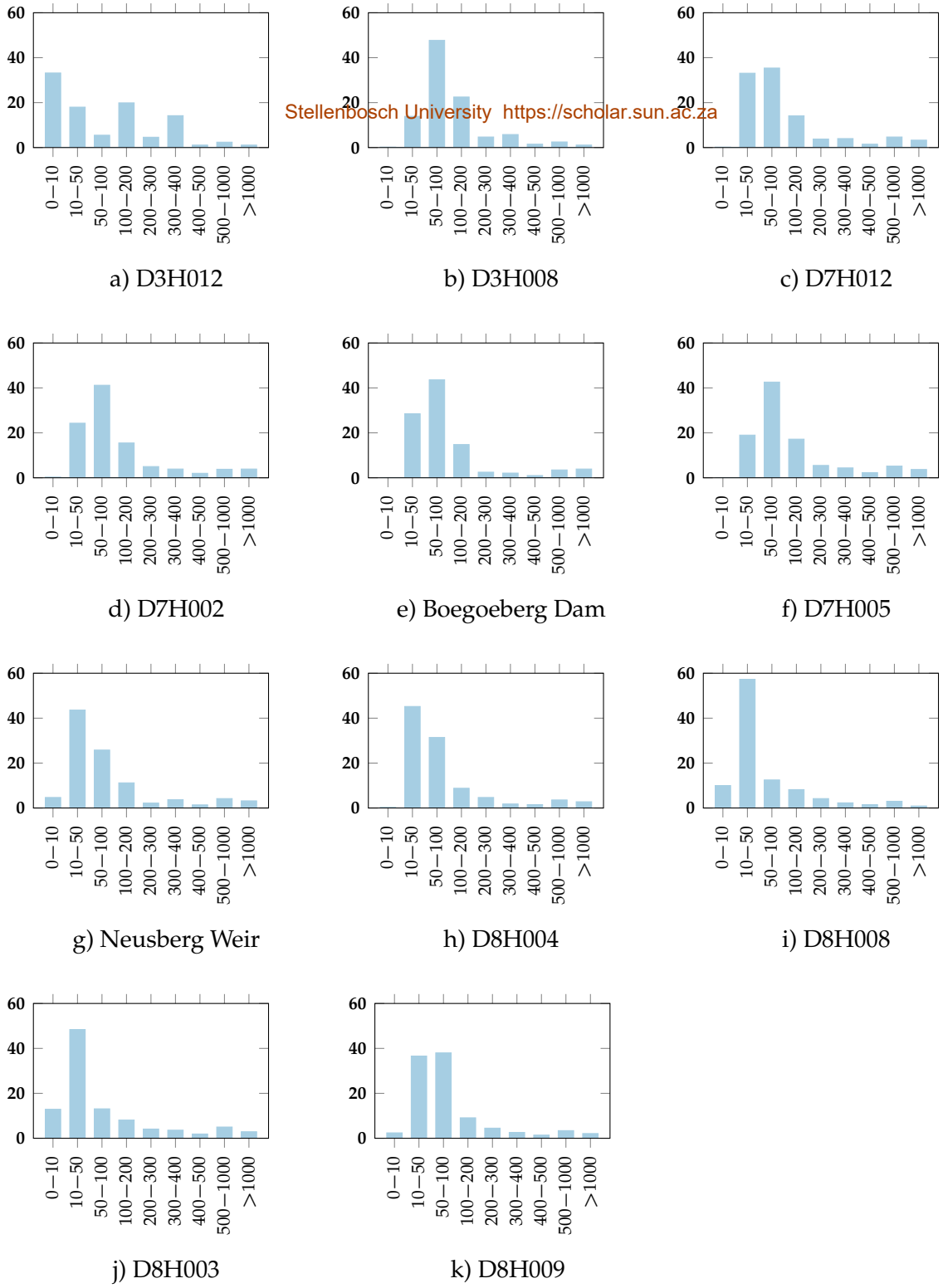
Assessing data availability across different flow regimes

The quantity of flow data across different flow regimes (i.e. low or high flows) can have an effect on how well the model can be trained for each of these flow regimes. We assess the availability of data across different flow regimes by plotting histograms of percentage of flow within each flow range in Figure 4.5 and Figure 4.6 for the Vaal and Orange River stations respectively. It is clear from inspecting the histograms that the majority of training data is for flows below 200 m^3/s for stations along the Orange River, and below 50 m^3/s for stations on the Vaal, Modder and Riet Rivers. It is to be expected since the flows only exceed these limits when the rivers are in flood. The availability of training data for the high flow regime is not a concern since the purpose of the model would be to optimise releases from Vanderkloof Dam during normal operating scenarios, not during flood events.



a) C9H021 b) C5H039 c) C5H049

Figure 4.5: Histograms of recorded flow along Modder, Riet and Vaal Rivers. Labels are station codes, units are percentage occurrence (%) for vertical axis and cubic metres per second (m^3/s) for horizontal axis.



a) D3H012 b) D3H008 c) D7H012

d) D7H002 e) Boegoeberg Dam f) D7H005

g) Neusberg Weir h) D8H004 i) D8H008

j) D8H003 k) D8H009

Figure 4.6: Histograms of recorded flow along Orange River. Labels are station codes, units are percentage occurrence (%) for vertical axis and cubic metres per second (m^3/s) for horizontal axis. Stations are ordered according to chainage distance from Vanderkloof Dam. Flows at D7H008 and D7H017 were combined to calculate flow at Boegoeberg Dam. Flows at D7H014, D7H015 and D7H016 were combined to calculate flow at Neusberg Weir.

Identifying trends and inconsistencies

There are three peaks in the histogram for Dooren Kuilen (D3H012), displayed in Figure 4.6, that correspond to water being released through no, one or two hydro-power turbines respectively. It can be seen that these peaks are attenuated by the time the water reaches the next station, Marksdrift (D3H008), where the distribution is much smoother with a single peak for the 50 m^3/s to 100 m^3/s category.

Considering the distributions of flows for the stations one would expect the stations closer to the Vanderkloof Dam to have a higher percentage occurrence in the higher flow categories and then a gradual shift of the distribution towards the lower categories as it progresses downstream. This trend is observed from Dooren Kuilen (D3H012) to Marksdrift (D3H008), with Dooren Kuilen exhibiting a high percentage of flows for 100 m^3/s to 200 m^3/s and 300 m^3/s to 400 m^3/s and Marksdrift a high percentage of flows for 50 m^3/s to 100 m^3/s. This trend is, however, not consistent across all stations; for instance the more downstream of the two stations at Vioolsdrif (D8H009) has a higher percentage of flows for 50 m^3/s to 100 m^3/s than the more upstream station (D8H003). This does not seem reasonable since these stations are only 15 km apart, yet there is a difference of more than 20 % for that category. Fortunately the station at Vioolsdrif (D8H003) does not display the same characteristics and will be considered in the rest of this thesis as the most downstream station on the Orange River.

Although there are some stations which are known to produce less accurate measurements for some flow scenarios (Le Grange et al., 2009), this should not have a dramatic effect on the training of an ANN model. For a flow routing model to be of use for the DWS, it merely has to predict what a downstream station would measure the flow as, even if the measurement is inaccurate. We suggest that a DWS employee would be able to manage releases from the Vanderkloof Dam effectively by only knowing what the resulting downstream station measurements would be. This subtly different objective allows us to simplify the flow routing problem since we do not need to account for inaccuracies of flow measurements at specific stations.

Assessment of sampling rate

We assess the sampling rate over time by plotting the average number of measurements taken per year in Figure 4.7. This plot indicates how the average sampling rate at the flow gauging stations increased from approximately 500 /year ($\pm$2 /day) in 1970, to 2 000 /year ($\pm$5 /day) in 1980 and then to more than 30 000 /year ($\pm$80 /day) in 2010. Closer inspection of recent measurements reveals that all flow gauging stations currently take measurements at 12 minute intervals (120 /day) when they are functioning correctly.

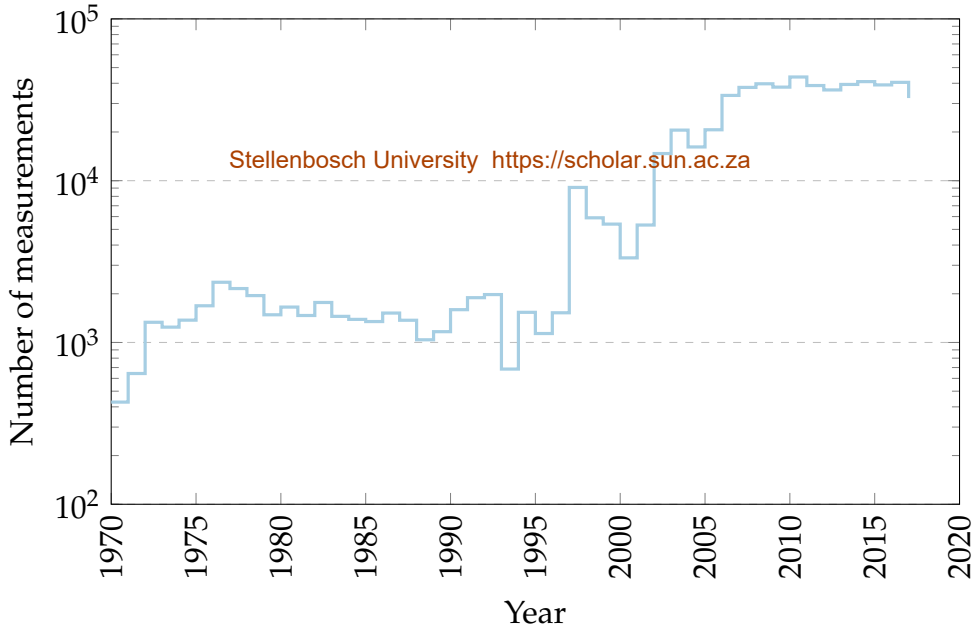


Figure 4.7: Average number of measurements taken per year across all flow gauging stations on Orange River.

Surprisingly, this increase in data resolution does not have a dramatic effect on the accuracy of measurements since there are rarely dramatic variations in flow, other than at Dooren Kuilen (D3H012) just downstream of the dam. By the time the water reaches the next station at Marksdrift (D3H008) 174 km downstream, the peaks in flow have largely been attenuated. This can be seen in Figure 4.8. This attenuation of peak flows means that recording flow at a higher rate does not necessarily improve accuracy significantly, since changes in flow occur slowly and are captured well enough with a lower sampling rate. For the station at Dooren Kuilen, however, a higher sampling rate would make a difference to the accuracy of the data. Fortunately this station has recorded measurements at a relatively high rate ($\pm$15 /day) since its comissioning in 1981, which captured releases from the Vanderkloof Dam with sufficient accuracy.

In terms of meteorological stations, all of the stations that were considered in this thesis collect daily samples and have not varied this sampling rate since 1970.

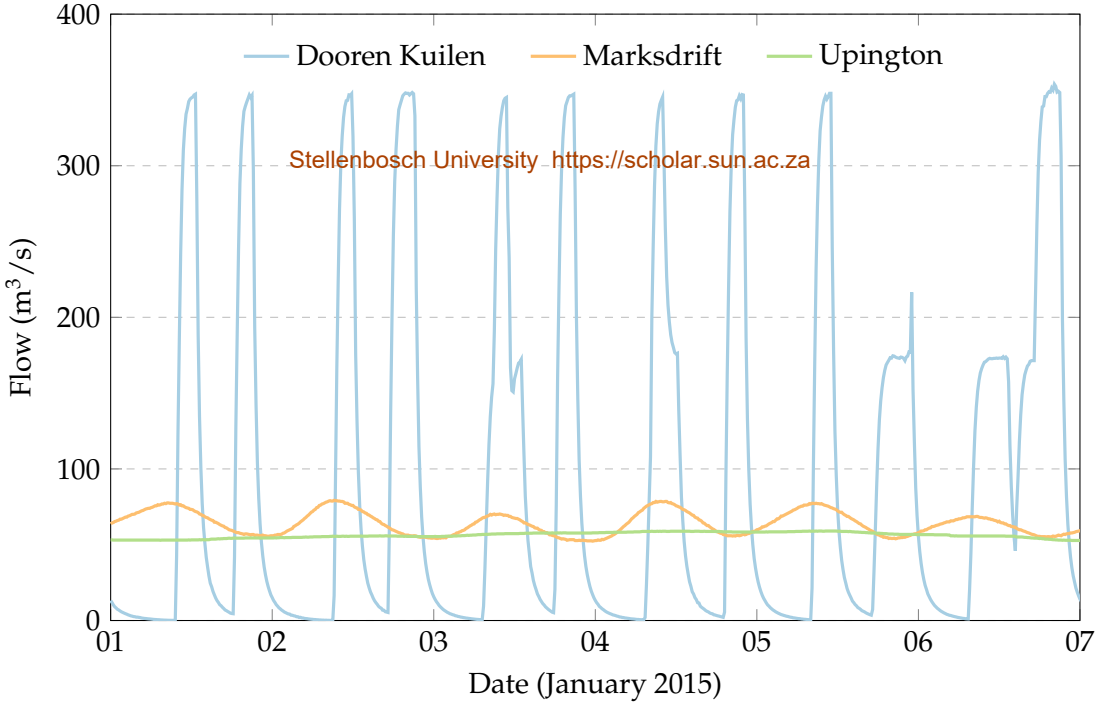


Figure 4.8: Recorded flow at Dooren Kuilen, Marksdrift and Upington illustrating the attenuation of peaks in flow. These stations are located just downstream, 174 km downstream and 635 km downstream of Vanderkloof Dam respectively.

4.3 Preparation of training samples

Training an ANN requires a set of input-output samples. This set of data is called a training set and needs to be constructed from the raw data that was collected for the different flow gauging and meteorological stations.

Constructing a training set involves:

1. re-sampling data at consistent intervals,
2. deciding on an input time window,
3. removing training samples with missing data,
4. adjusting the distribution of data by normalising it, and
5. splitting the data into different parts that will be used for training, validation and final testing.

Each of the steps above will be discussed in the following sections.

Re-sampling data at consistent intervals

Since the sampling rate at flow gauging stations changed over time (see Figure 4.7), the raw input data needs to be re-sampled at consistent intervals. ANNs require input that are at consistent intervals to train and produce reasonable output. Although the stations currently all have a sampling rate of 12 minutes, this high sampling rate is not required to capture the changes in flow along the river. Re-sampling the data at an interval of one hour was found to capture changes in flow sufficiently, even at the station displaying the most rapid changes in flow, i.e. Dooren Kuilen (D3H012), just downstream of Vanderkloof Dam. We can confirm that this is the case by viewing the smooth transitions of flow in Figure 4.8, which was sampled at one-hour intervals. Re-sampling the data at a lower rate also has the side-effect of lowering the number of input parameters (i.e. the number of nodes in the first layer) of the ANN, which would lead to lower training times and computer hardware memory usage (refer to Section 3.2).

A training sample consists of an input-output pair. The input part would be a vector containing the recorded flows $\mathbf{x}$ at an upstream station, starting at time-step i, for n consecutive time-steps. The output value would be the resulting flow y_{i+n} at a downstream station at time-step $i+n$. With x_i representing the flow at time-step i, training samples will take the form:

$$\begin{bmatrix} x_i & x_{i+1} & x_{i+2} & \cdots & x_{i+n} \end{bmatrix} \mapsto y_{i+n}. \tag{4.1}$$

Although the flow recorded at the upstream station at time-step $i+n$ would not have any effect on the downstream flow at that same instant, we did not offset the input time window to account for the lag between the stations. Instead we hypothesised that the models would learn to ignore the input parameters that do not have any effect on the output.

For meteorological stations, which collect data at daily intervals, the data was effectively up-sampled to hourly intervals to match the flow measurement sampling rate, since the two data streams were combined in each input sample. With $\mathbf{x}$ representing flow data and $\mathbf{e}$ representing evaporation data, training samples would take the form:

$$\begin{bmatrix} x_i & x_{i+1} & x_{i+2} & \cdots & x_{i+n} \\ e_i & e_{i+1} & e_{i+2} & \cdots & e_{i+n} \end{bmatrix} \mapsto y_{i+n}. \tag{4.2}$$

Choosing an input time window

When constructing a training sample, we need to decide on the length of the input time window (n). If, for instance, we use the preceding year's worth of data at an upstream station to predict the flow at a single time step at a downstream location, this would require a model with a very large number of parameters in the input layer (approximately 36 000 nodes), resulting in long training times and possibly no improvement in the accuracy of the resulting model. Reducing the input time window to as small a range as possible would yield significant speed-ups in training time.

The time it takes for water to propagate from an upstream station (where input measurements are recorded) to a downstream station (where flow is predicted) may be a good approximation of input time window which is required for training data. As a rule of thumb, the rate of travel of water along the Orange River downstream of the Vanderkloof Dam is rarely less than 100 km /day. Based on this rate of travel, one would need approximately two days worth of input data to predict the flow at the output station Marksdrift (D8H003), 174 km downstream of the input station Dooren Kuilen (D3H012). Since lower flows translate to a lower rate of travel, we propose to double this input time window and choose a 4 day window as input (corresponding to 96 time steps). This means the input layer for a network taking these samples as input would require only 96 nodes. We will confirm this approach in Chapter 5 where we will investigate the effect of the input time window on training time and accuracy.

Removing training samples with missing data

A neural network requires values for all input and output parameters for training and therefore we need to exclude all training samples that do not have data for any input or output value. Since we re-sample the raw data to hourly values using interpolation, we would require every re-sampled value to have an actual measured value in relative close proximity. Based on the rate of change in flow and evaporation we observe in the raw data, we consider the following interpolated values invalid:

- interpolated flow values that do not have a measured value within a day on either side, and
- interpolated evaporation values that do not have a measured value within three days on either side.

After we identify these invalid values, we remove all training samples that contain them from our training data set.

Normalising data

The training process has been found to converge faster when data is normalised (Ioffe and Szegedy, 2015). For the purpose of this thesis, we apply feature scaling, which involves scaling the data to the range $[0, 1]$ by applying the following function to each input (x_i) and output ($\hat{y}_i$) value:

$$k_{\text{normalised}} = \frac{k - \min(\mathbf{k})}{\max(\mathbf{k}) - \min(\mathbf{k})}. \tag{4.3}$$

Feature scaling would cause all inputs, irrespective of their original distribution to be within the same range. This means even if flow measurements range from $0\,\text{m}^3/\text{s}$ to $10\,000\,\text{m}^3/\text{s}$ and evaporation from 0 mm to 30 mm, the resulting inputs would all be in the range 0 to 1. This adjustment assists a model to not bias one variable above another just because the original ranges of the data sets were different. Note that the minimum and maximum values for the data sets are determined at training time, and are also used to scale the data before prediction at test time.

Splitting data into training, validation and test sets

While training an ANN we continually assess the value of the loss function for each complete pass through the data set, or epoch. We know that while the loss decreases, the model is approaching a minimum of the loss function. This decrease in the loss function does not, however, protect us against overfitting on the training data and could cause the model to produce less accurate predictions for data outside of the training set. To counter this, we set aside a portion of the data before training which will not be used in gradient descent optimisation, called the validation set. We calculate the loss function for this portion of the data during each epoch of training, and plot these values along with the training loss on a graph. Once the loss for the validation set shows an increasing trend, even though the loss for training set may be decreasing, the model is overfitting the training data.

By re-using the validation set repeatedly to assess different model architectures or hyper-parameter values, we are also in a way using it for training our models. In order to have an unbiased measure of accuracy, we set aside another portion of the data, called the test set. This data set should only be used as a final sense-check before results are published and should not inform any of the model architecture or hyper-parameter choices.

The proportional split between training, validation and test data was obtained with a percentage split of 60–20–20. Typically this split is obtained by randomly picking samples from the entire data set and assigning them to each of the sets. In the case

of the input data used in this thesis there is some overlap between consecutive data samples, which would make random assignment of samples to these sets unfeasible. If we were to randomly assign two consecutive (very similar) training samples to the training and validation sets respectively, we would essentially be tainting our validation set with training data. To illustrate this, consider two training samples for y_{i+n} and y_{i+n+1} respectively:

$$\begin{bmatrix} x_i & x_{i+1} & x_{i+2} & \cdots & x_{i+n} \end{bmatrix} \mapsto y_{i+n}, \tag{4.4}$$

$$\begin{bmatrix} x_{i+1} & x_{i+2} & \cdots & x_{i+n} & x_{i+n+1} \end{bmatrix} \mapsto y_{i+n+1}. \tag{4.5}$$

Comparing these two training samples reveals that the input vectors contain identical terms, x_{i+1} through to x_{i+n}. Including one of these samples as a training sample and the other as a validation sample would reduce the potency of the validation set to identify overfitting and to act as a measure of how well the model generalises to non-training data. We will counter this by, rather than randomly assigning samples to sets, ordering our data by time and assigning the first 60% to the training set, the next 20% to the test set and the final 20% to the validation set. In this way there would be no overlap between the different sets, other than a small number of samples close to the transitions between them. Because we assign the sets in this order, we essentially train the models on older data and test the model against newer data, which corresponds to what will happen in an operational setting.

4.4 Influence of data on model application

Model limitations

The models which we will train make use of historical data recorded at fixed locations along the river. Because of this, and the nature of the data, these models would intrinsically have a number of limitations:

- Flows at locations other than the flow gauging stations cannot be predicted, since the models can only be trained to predict flow at locations where flow data is available.
- Flow regimes for which there are no or very little recorded data, for instance a flood with an unprecedented magnitude, would not be predicted with a high degree of accuracy.

Although these limitations should be considered, optimising releases from the Vanderkloof Dam would not require detailed flow predictions at every position along the river and would typically only need to predict flows within the normal operating flow regime, for which there is sufficient training data.

Restrictions on data availability for models run in operational settings

A subtle difference in handling flow data and evaporation is that flow in the river is controlled by releases from the Vanderkloof Dam, which can be known ahead of time, whilst evaporation values cannot be predicted in the same way. This means that we can include future releases from Vanderkloof Dam as input into an optimisation model, but we would not have access to future evaporation in the same way. To train a model that can be run in practice one would need to exclude any evaporation data during training time that would not be available in an operational setting.

4.5 Summary

The data the DWS has collected along the river reaches investigated in this thesis is of sufficient quantity and quality to enable the training of an ANN. Although there may be other data sources available, such as weather predictions and aerial photographs of irrigated areas, we limit ourselves to the data the DWS has available in an operational setting. This would allow the models we produce to be used in practice.

Chapter 5

Application of deep learning to flow routing

Finding a suitable neural network architecture and set of hyper-parameters that apply well to a specific problem can be a hit-and-miss exercise. We approach this problem systematically by:

- establishing baseline accuracy measurements by applying simple linear regression,
- setting up a variety of models for three neural network architectures: fully-connected networks, CNNs and RNNs, and
- experimenting with different hyper-parameter variations for each of the models.

Once the most promising model for each of the architectures has been identified, additional training time is assigned to train the model to convergence. Finally, we compare the results from the different model architectures to each other and select the one used for subsequent experiments in this thesis. For the purpose of comparison, all the experiments conducted in this chapter model the 174 km river reach between the Dooren Kuilen (D3H012) and Marksdrift (D3H008) gauging stations.

5.1 Software and hardware used

The Python programming language (version 3.6.2) and the TensorFlow library (version 1.0.3) were used for all experiments conducted in this thesis. The code repository, with the relevant data sets and code for all models, is hosted on GitHub (Briers, 2018). Because the data sets are small, compared to image data sets, models could be trained using a GeForce GTX 1050 Ti GPU.

5.2 Measures of model performance

We use three measures of model accuracy throughout this chapter:

- mean absolute percentage error (MAPE)

$$\text{MAPE} = \frac{100}{N} \sum_{n=1}^{N} \mid \frac{y_n - \hat{y}_n}{y_n} \mid, \tag{5.1}$$

- root mean-squared error (RMSE)

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{n=1}^{N} (y_n - \hat{y}_n)^2}, \tag{5.2}$$

- coefficient of determination (R^2)

$$\text{R}^2 = 1 - \frac{\text{RMSE}}{\sum_{n=1}^{N} (y_n - \bar{y})^2}, \text{ with } \bar{y} = \frac{1}{N} \sum_{n=1}^{N} y_n, \tag{5.3}$$

with $\hat{y}_n$ representing the flow rate predicted by the model for sample n, y_n the recorded flow rate and N the number of samples.

All models will be compared against these three measures, but preference will be given to MAPE since it penalises errors across all flow regimes inversely proportional to the magnitude of the flow rate. RMSE and R^2 would give more weight to larger errors that typically occur at high flow regimes, but which represent small errors when considered as percentages. This makes RMSE and R^2 less suited as measures of accuracy for a model which would be used in low flow (lower than 200m^3/s) operational settings. Lower MAPE and RMSE values both signify better accuracy. R^2 is the proportion of the variation the model explains, and a value closer to 1 indicates improvement in accuracy.

Another measure of model performance which we will assess is the accurate timing of peaks and troughs in the flow rate. Although it is hard to quantify this metric, we will assess it by visually inspecting how well predictions match increases and decreases in flow rate across the validation data set. This measurement is especially important when building a model that could be used to optimise releases from the Vanderkloof Dam, since the timing of water arriving downstream is critical when informing such decisions.

5.3 Determining the input time window

Before constructing a training data set for our models we need to decide on a time window for the input data recorded at the upstream station. In Section 4.3 we propose using a four-day window at the upstream Dooren Kuilen station, preceding the output time-step we are predicting for at the downstream Marksdrift station. In order to confirm if this input time window is appropriate, we examine a specific instance where releases from the Vanderkloof Dam was increased and identify the resulting downstream peak in flow rate. Figure 5.1 shows three days of increased releases from the Vanderkloof Dam measured at Dooren Kuilen from 20 to 22 July 2013 and the corresponding peak in flow rate at the Marksdrift station. The delay between the increased releases and the increase in flow rate at the downstream station gives an indication of the rate at which water propagates down the Orange River.

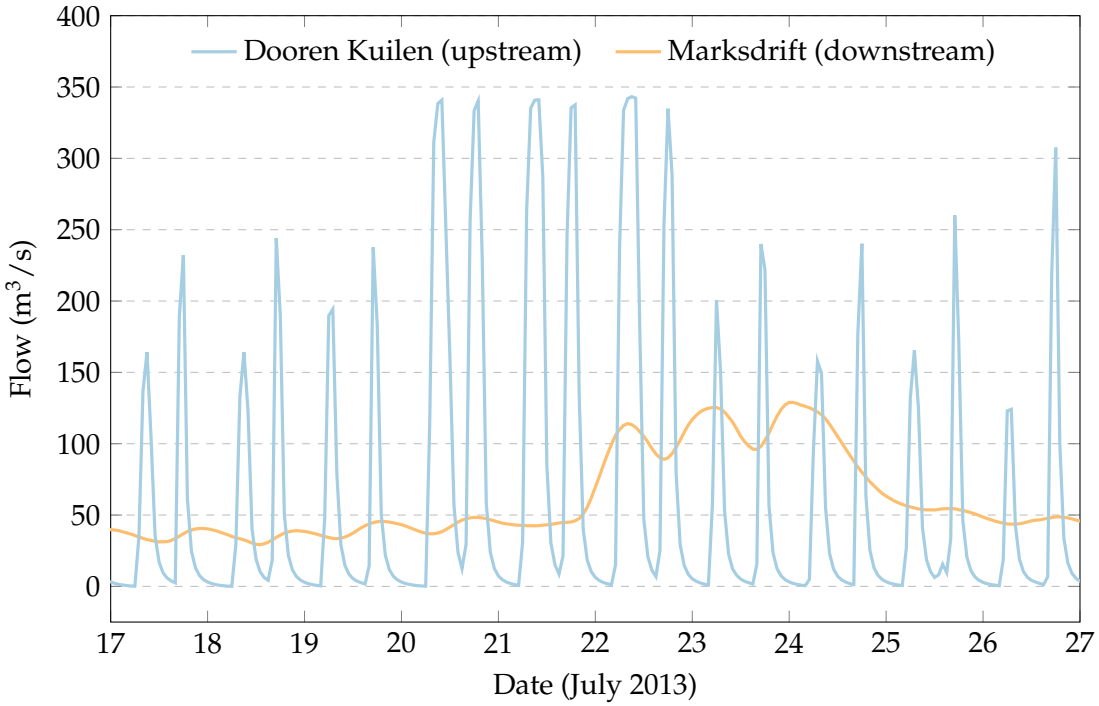


Figure 5.1: Comparing recorded flow rate at Dooren Kuilen (D3H012) and Marksdrift (D3H008) to establish appropriate input time window.

From Figure 5.1 we see that increased releases from the Vanderkloof Dam reach Marksdrift station in less than two days, early on 22 July 2013, and that this elevated flow rate dissipates within three days of the last higher release from the dam, around 26 July 2013. To account for some variability in this rate of propagation, we will use an input time window of four days (or 96 hours) for this river reach.

5.4 Establishing a baseline

We set up a linear regression model to establish a baseline to measure our models against. Although this model is not expected to provide accurate flow rate predictions, we can compare the performance of the neural networks we set up against it. Since ANNs are particularly suited to problems where there are subtle non-linear relationships between input and output values, comparing our results with a purely linear model may give us an indication if this capability is being leveraged. When applying least-squares regression to the training data set, we obtain the results listed in Table 5.1.

Table 5.1: Linear regression results on the validation set.

Accuracy measure	Result
MAPE	15.0 %
RMSE	23.1 m^3/s
R^2	0.978

A sample of predictions from the validation data set for July 2013 is displayed in Figure 5.2. July 2013 was chosen for the purpose of visual comparison for all models since it contains flow rates in low and high flow regimes, and it is a time period for which none of the models set up in this chapter performed above their average performance.

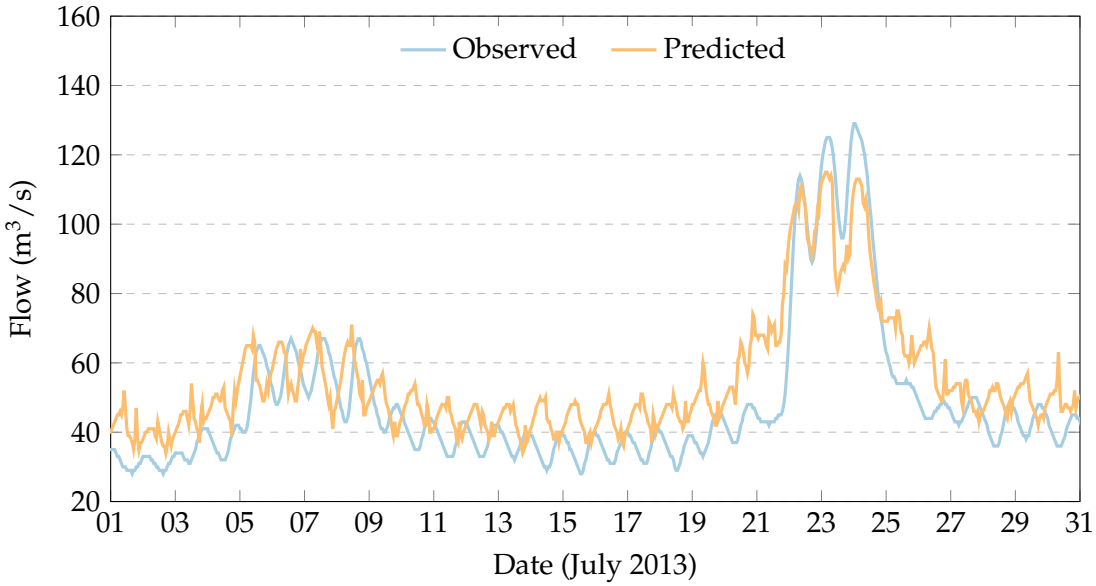


Figure 5.2: Sample of flow rate predictions at Marksdrift gauging station for the linear regression model.

Linear regression captures the trends in flow rate surprisingly well considering the simplicity of the model. Although there is a lot of noise in the predicted time series we can see that some of the peaks and troughs in the flow rate are well-matched, especially at flow rates above $80\,m^3/s$. At flow rates below $80\,m^3/s$, the timing of the predicted flow rate peaks are not synchronised with the recorded rates. These trends are observed, not just for this data sample, but across the validation data set.

5.5 Fully-connected neural networks

To efficiently identify a well-performing model configuration for a fully-connected neural network, we gradually increase the complexity of the model until such a point where adding additional layers or nodes per layer does not increase the accuracy of the model on the validation set. We first investigate the effects of widening the hidden layer of a single-layer network and then adding additional layers to the network.

Single-layer fully-connected network

We start with a neural network with a single hidden layer followed by a dropout layer. Other hyper-parameters are defined as in Table 5.2. We choose the linear identity function as the activation function of our final layer throughout this thesis, since we are predicting a continuous variable. We use a variation of stochastic gradient descent that applies momentum in the direction of descent, improving training stability and time to convergence (Sutskever et al., 2013).

Table 5.2: Hyper-parameters for fully-connected neural network.

Hyper-parameter	Value
Loss function	Mean-squared error
Learning rate	0.1
Mini-batch size	64
Activation function for input layer	ReLU
Activation function for hidden layer(s)	ReLU
Activation function for output layer	Identity function
Dropout rate for dropout layer	0.5
Optimiser	Stochastic gradient descent
Momentum factor	0.8

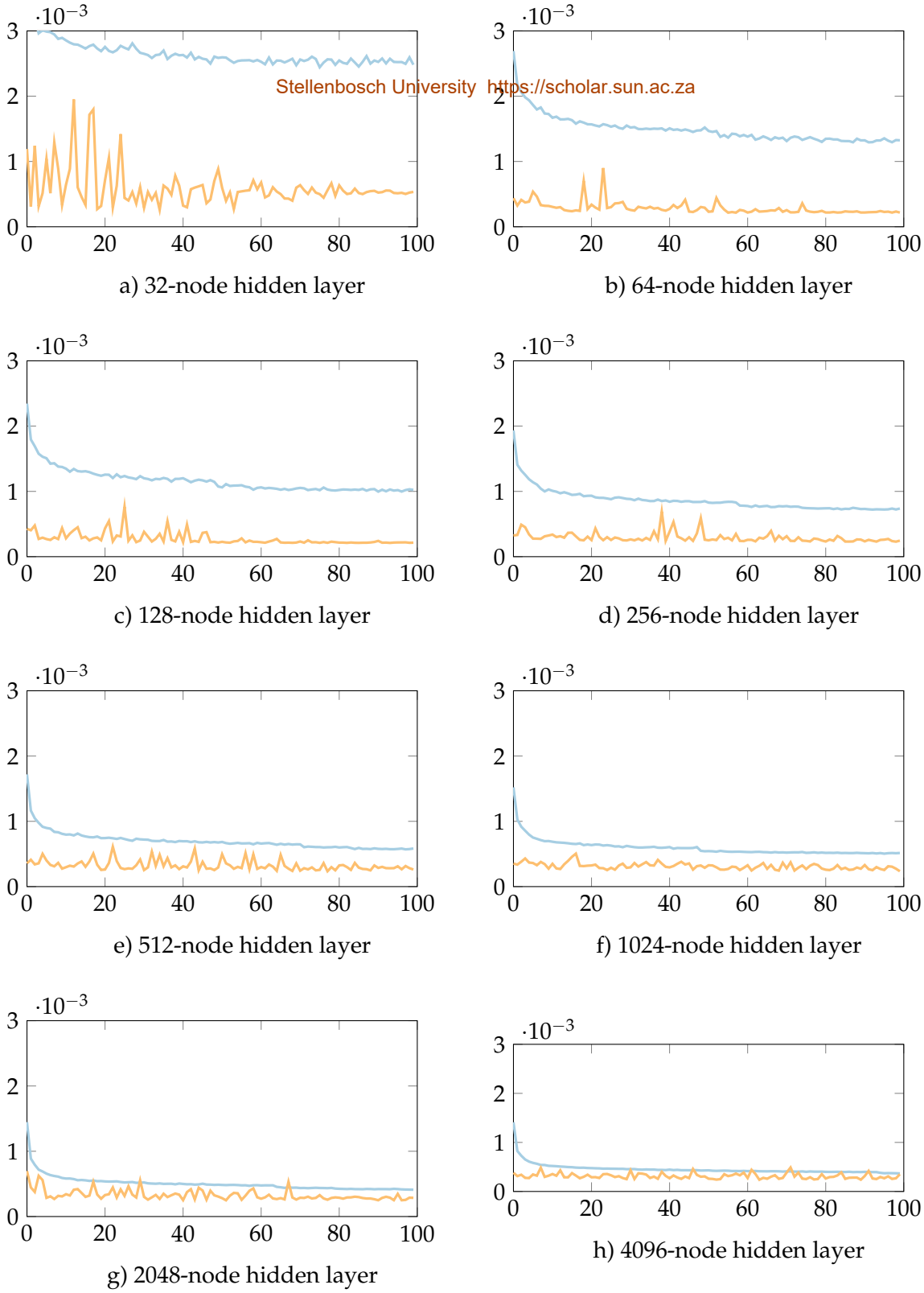
We now train a number of models varying the number of nodes in the hidden layer from 32 to 4096 to find the best performing single-layer model. Table 5.3 gives a comparative view of the performance of the single-layer models.

Table 5.3: Comparison of single-layer fully-connected network results on the validation set.

Number of nodes	MAPE	RMSE	R^2
32	19.1	34.0	0.952
64	17.3	21.7	0.981
128	15.9	20.7	0.982
256	14.6	22.5	0.979
512	15.5	22.5	0.979
1024	15.1	27.7	0.968
2048	15.4	23.5	0.977
4096	17.5	26.0	0.972

Figure 5.3 illustrates the training performance by plotting the training and validation loss per training epoch for each of the hidden layer sizes. It is apparent from this figure that there is an increase in accuracy as one increases the layer size from 32 to 64 nodes. We can see that the 32-node model has a higher loss and underfits the data compared to the other models. In general the training loss continues to decrease as we add more nodes, however, the validation loss plateaus and does not show improvement beyond 256 nodes in the hidden layer.

Training set loss is typically lower than validation loss during training of a neural network, since the accuracy on the validation set rarely surpasses accuracy on the training set. In Figure 5.3 we observe that this is not the case. This is caused by the non-random assignment of data samples to the training and validation sets which result in dissimilar data distributions, and the resulting lower validation set loss. It is hypothesised that the validation set included comparatively less high-flow events which resulted in a lower loss. A similar trend was observed for other models trained in this thesis.



a) 32-node hidden layer

b) 64-node hidden layer

c) 128-node hidden layer

d) 256-node hidden layer

e) 512-node hidden layer

f) 1024-node hidden layer

g) 2048-node hidden layer

h) 4096-node hidden layer

Figure 5.3: Training loss (blue) and validation loss (orange) for single hidden layer fully-connected neural networks with varying number of nodes over 100 training epochs.

Training best performing model to convergence

The models which were trained for the comparison in Table 5.3 have not converged after 100 epochs, since their training losses continued to decrease as training progressed. We undertake a longer training run of 1000 epochs for the most promising model, a single fully-connected network with one hidden layer of 256 nodes. The resulting model's training performance against the training and validation data sets are displayed in Figure 5.4.

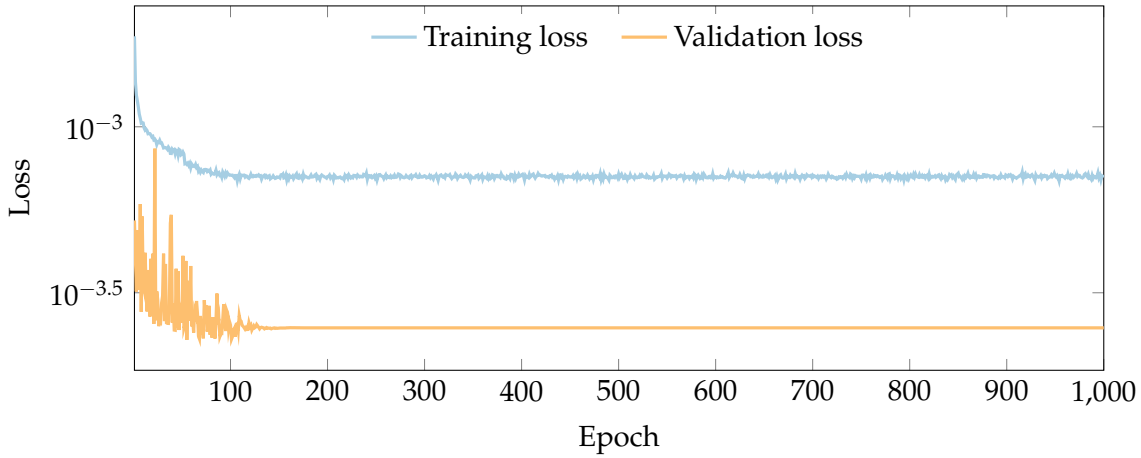


Figure 5.4: Training and validation loss when training best-performing single-layer fully-connected network to convergence.

Typically one would stop training once convergence is reached, around epoch 120, but here we continued training for the full 1000 epochs to illustrate that the model would not improve beyond a certain point. Increasing accuracy can then only be achieved by changing model hyper-parameters or adding more training data. The results for the network are listed in Table 5.4.

Table 5.4: Validation results for best performing single-layer fully-connected ANN with a 256-node hidden layer.

Accuracy measure	Result
MAPE	15.5 %
RMSE	21.9 m^3/s
R^2	0.980

The result for the longer training-run shows worse MAPE values compared to the shorter run, but better RMSE and R^2 measures. This is not unexpected, since we use mean-squared error as our loss function. Additional training would conceivably make the model favour introducing errors at lower flow regimes rather than high flow

regimes, which would improve the RMSE measure but negatively affect the MAPE measure.

Selecting a more suitable loss function

To counter the tendency of the mean-squared error loss function to negatively impact the MAPE accuracy, we train another two models using the alternative loss functions listed in Section 3.2, namely the mean-squared logarithmic error and the logarithm of the hyperbolic cosine of the error. Although using MAPE may seem like a natural choice as a loss function, it led to instabilities during training, possibly because the loss would fluctuate dramatically for errors at lower flow rates. The results for the two models trained to convergence over 300 epochs, using the alternative loss functions, are listed in Table 5.5. The results for the model trained with mean-squared error loss function are also included for comparison.

Table 5.5: Comparing validation results using different loss functions for a single-layer fully-connected network.

Loss function	MAPE	RMSE	R^2
Mean-squared error	15.5	21.9	0.980
Mean-squared logarithmic error	15.0	23.6	0.977
Logarithm of the hyperbolic cosine of the error	15.1	21.9	0.980

Table 5.5 indicates that although the differences between using the different loss functions are small, using the logarithm of the hyperbolic cosine of the error does seem to improve the MAPE without sacrificing accuracy of the other two measures. For this reason we will continue to use this loss function for further experiments in this chapter.

Single-layer fully-connected network results

A sample of flow rate predictions at Marksdrift for July 2013 is displayed in Figure 5.5. Similar to the regression model, predictions for peaks and troughs at higher flows (above 80 m^3/s) are matched well in terms of timing and magnitude. Predictions for lower flows do not exhibit the same accuracy as those for higher flows. This trend is observed across the validation data set, with both higher and lower-than-observed predictions being produced.

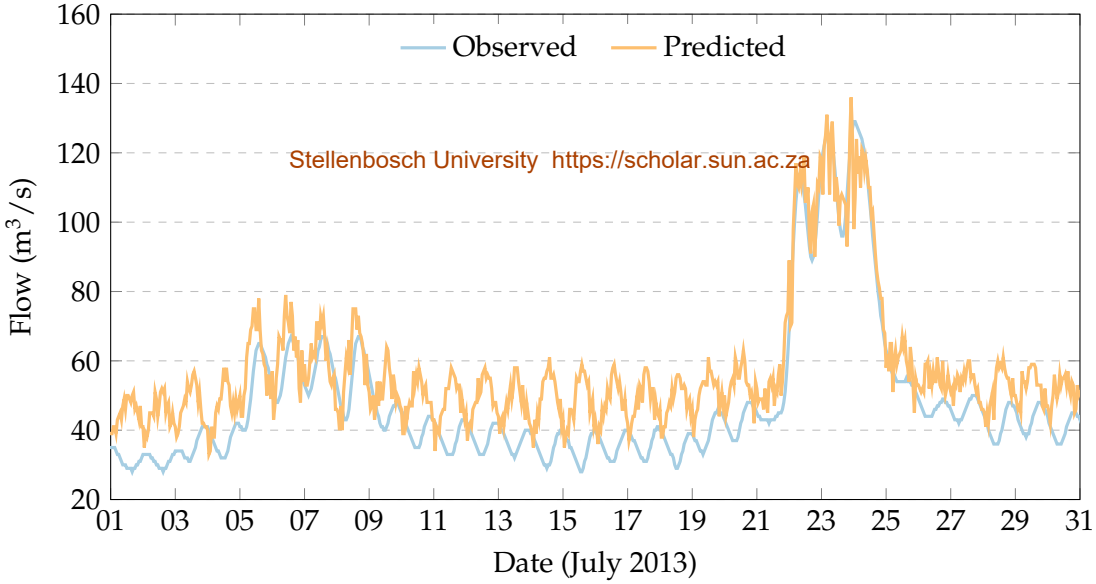


Figure 5.5: Sample flow rate predictions at Marksdrift station for best performing single-layer network.

Multiple layer fully-connected networks

From the results in Table 5.3 it does not seem as if additional model complexity would increase the accuracy, but to confirm this, we also run experiments for two, three, four and five layer fully-connected neural networks. For these models we use 256 nodes per layer and identical hyper-parameters to the ones used for the single-layer networks. The results for each of the multi-layer models, which were trained to convergence over 250 epochs, are given in Table 5.6. The results for the single-layer 256 node network are also included in this table for comparison.

Table 5.6: Comparing validation set multi-layer fully-connected network results.

Number of layers	MAPE	RMSE	R^2
1	15.1	21.9	0.980
2	15.0	21.3	0.981
3	15.3	21.2	0.981
4	15.5	21.9	0.980
5	15.5	22.0	0.980

From Table 5.6 we see that the two-layer network shows a slight increase in performance across all measures. A sample of flow predictions for July 2013 for Marksdrift are displayed in Figure 5.6.

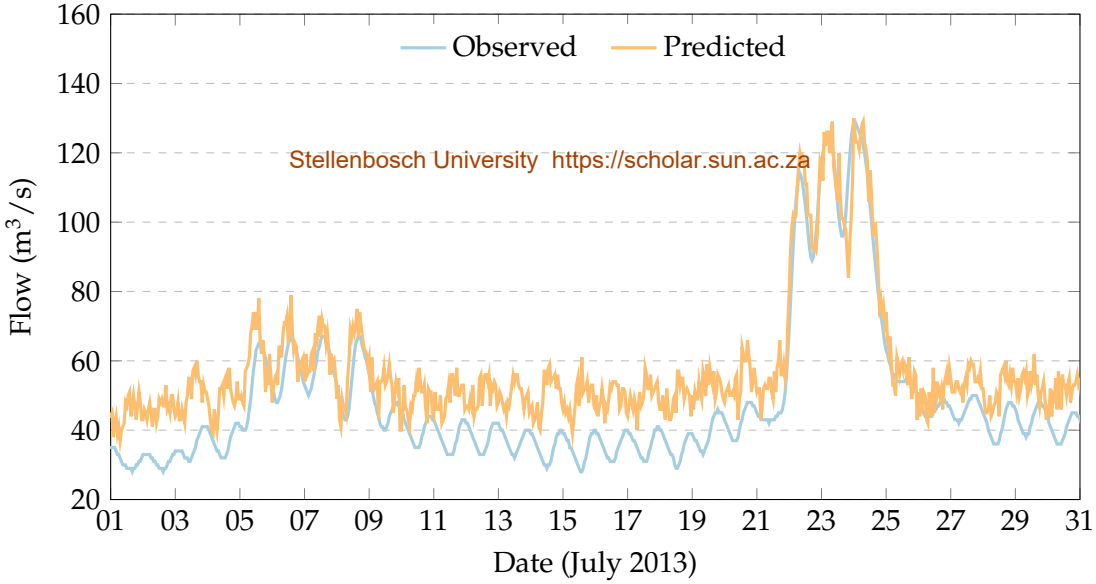


Figure 5.6: Sample flow rate predictions at Marksdrift station for two-layer fully-connected network .

Interestingly the timing for predictions of flow peaks and troughs at low flows (below 80 m^3/s) has improved even though the accuracy of these prediction are visibly lower than the predictions at higher flow rates. This seems to indicate that the additional layers improve the model's ability to account for variable attenuation across different flow regimes.

5.6 Convolutional neural networks

CNNs have proven to be effective at a range of applications such as image recognition and other problem domains, such as sequence modelling, which is typically associated with recurrent networks (Bai et al., 2018). Taking inspiration from the network architecture of the models developed by the Visual Geometry Group (VGG) (Simonyan and Zisserman, 2014) in their 2014 ImageNet submission, we experiment with four incrementally deeper CNNs. The network architectures for the networks are illustrated in Figures 5.7 to 5.10 and are code-named CNN4, CNN6, CNN8 and CNN10 respectively, based on the number of convolutional layers each architecture contains. The layers in Figures 5.7 to 5.10 are defined as follows:

- (x) 96-element input vector.
- (A) 2 convolutional layers with 64 filters of size 3. Convolution stride is 1. Output is 64 96-element vectors.

- (B) 2 convolutional layers with 128 filters of size 3. Convolution stride is 2 for the first layer and 1 for the second layer. Output is 128 48-element vectors.
- (C) 2 convolutional layers with 256 filters of size 3. Convolution stride is 2 for the first layer and 1 for the second layer. Output is 256 24-element vectors.
- (D) 2 convolutional layers with 512 filters of size 3. Convolution stride is 2 for the first layer and 1 for the second layer. Output is 512 12-element vectors.
- (E) 2 convolutional layers with 512 filters of size 3. Convolution stride is 2 for the first layer and 1 for the second layer. Output is 512 6-element vectors.
- (F) Fully-connected layer using ReLU activation function. Output is 2048-element vector.
- ($\hat{y}$) Fully-connected layer using linear activation function. Output is a single scalar value.

Other hyper-parameter values are indicated in Table 5.7. A dropout layer (not displayed on the diagrams) is included after fully-connected layer.

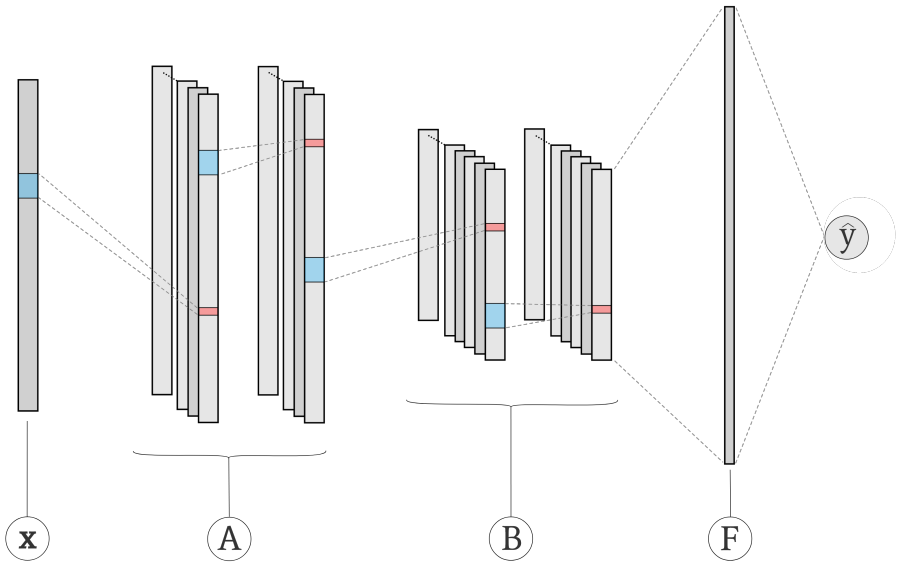


Figure 5.7: CNN4 convolutional neural network architecture.

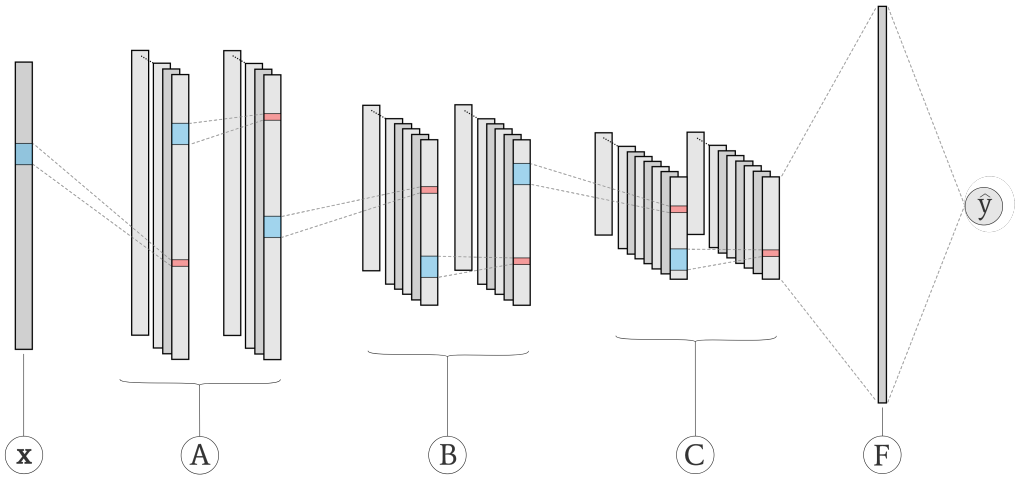


Figure 5.8: CNN6 convolutional neural network architecture.

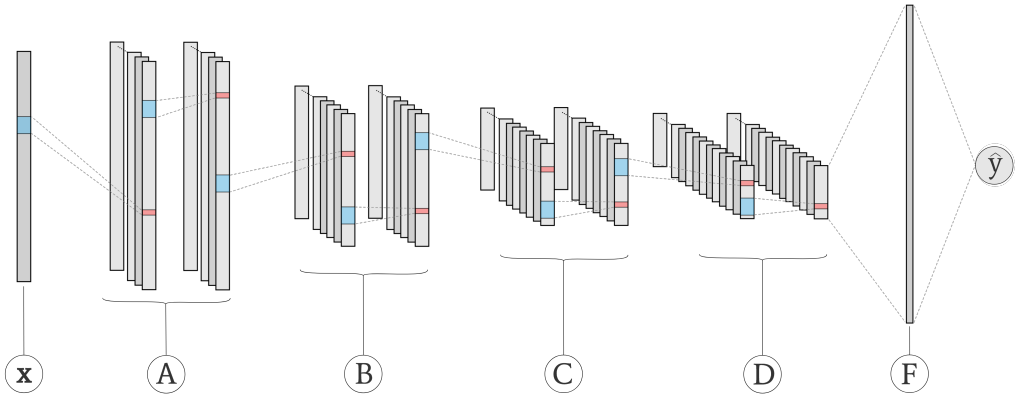


Figure 5.9: CNN8 convolutional neural network architecture.

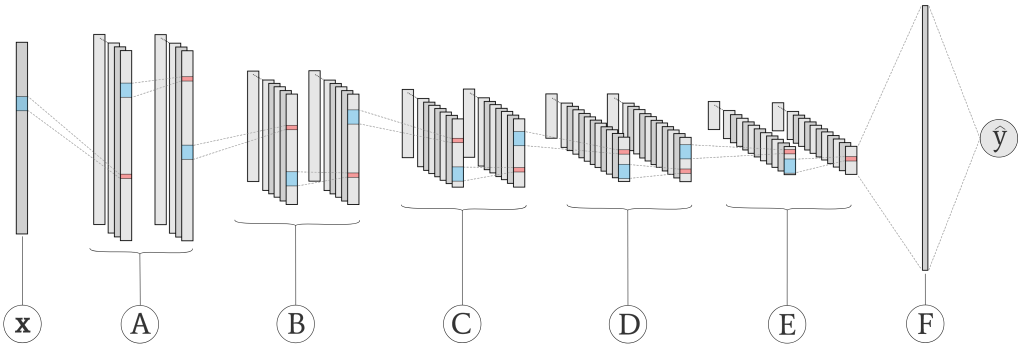


Figure 5.10: CNN10 convolutional neural network architecture.

Table 5.7: Hyper-parameters for CNN models.

Hyper-parameter	Value
Loss function	Logarithm of the hyperbolic cosine
Learning rate	0.01
Mini-batch size	64
Activation function for convolutional layers	ReLU
Activation function for output layer	Identity function
Dropout rate for dropout layer	0.5
Optimiser	Stochastic gradient descent
Momentum factor	0.8

Training these models for 100 epochs each yields the validation results listed in Table 5.8.

Table 5.8: Comparison of CNNs with variable number of convolutional layers.

Model code	MAPE	RMSE	R^2
CNN4	14.5	19.8	0.984
CNN6	21.6	25.3	0.973
CNN8	16.9	24.9	0.974
CNN10	13.9	21.1	0.982

The CNN10 model produced the lowest MAPE measure and was selected for a longer training period of 1000 epochs. The training and validation loss performance are plotted against training epoch (up to epoch 500) in Figure 5.11. Convergence is reached after approximately 400 epochs. Results from the converged model are given in Table 5.9. A sample of the predictions for July 2013 is displayed in Figure 5.12.

Table 5.9: Validation results for best performing CNN, CNN10 model with 10 convolutional layers.

Accuracy measure	Result
MAPE	13.6 %
RMSE	22.6 m^3/s
R^2	0.979

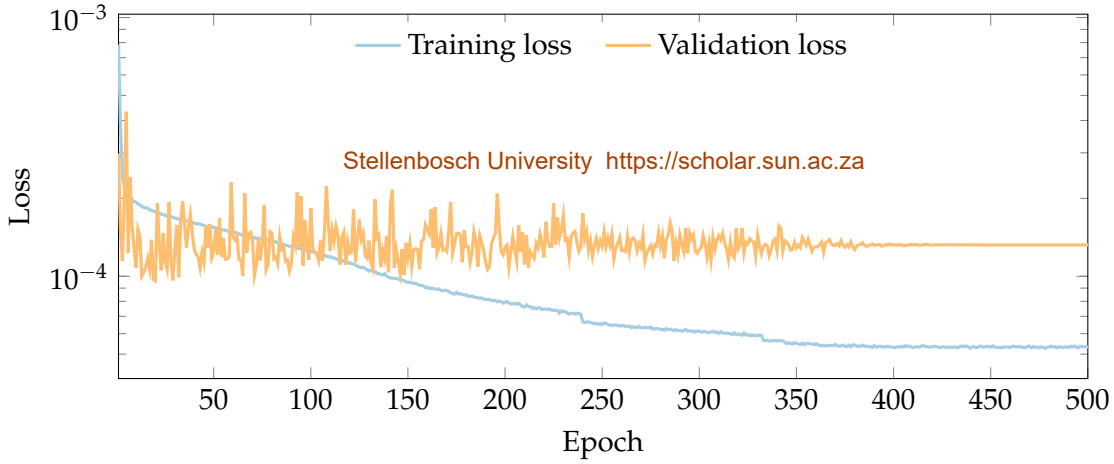


Figure 5.11: Training and validation loss when training best-performing CNN to convergence.

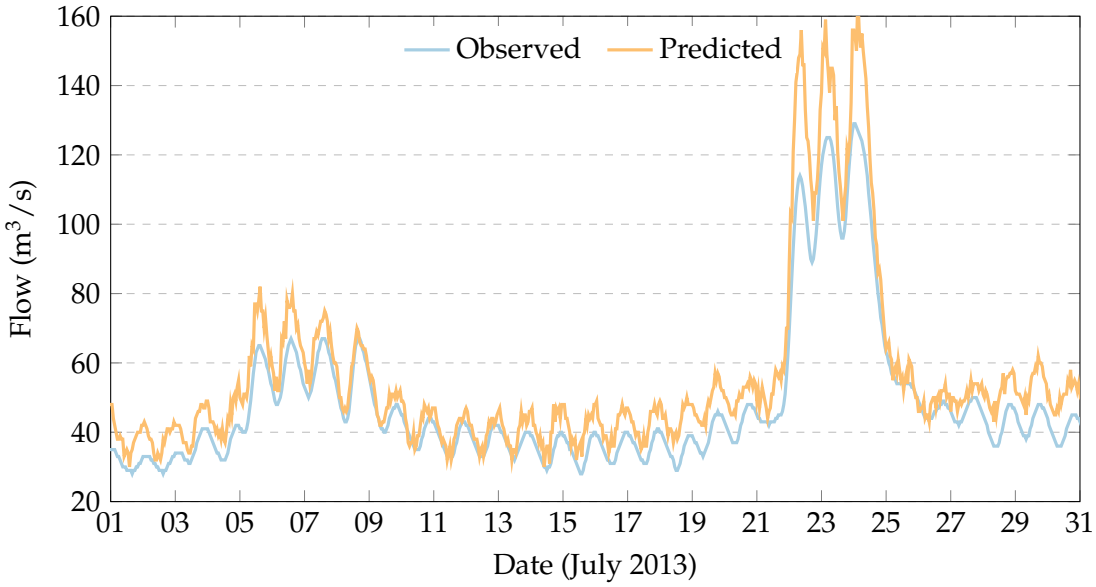


Figure 5.12: Sample flow rate predictions at Marksdrift station for best performing CNN.

After additional training the model showed a slight improvement in the MAPE measurement and a slight deterioration in the RMSE and R^2 measures. There is a slight upward trend in the validation loss in Figure 5.11, indicating that there may be some overfitting taking place. Analysing the results from Figure 5.12 reveals that the timing of peaks and troughs for high and low flow regimes are matched well. The predicted flow for the high-flow regime (above $80\,m^3/s$) is significantly overestimated for this specific data sample, but when inspecting the rest of the validation data set, this overestimation is not a consistent trend.

5.7 Recurrent neural networks

As with fully-connected networks and CNNs, we also set up a number of RNN models. We use RNNs with GRU layers exclusively because of the simpler internal structure and reduced training times they have, compared to LSTM layers. We use a single recurrent layer and vary the output dimension of the layer for our experiments. We follow the recurrent layer with a dropout layer to prevent overfitting. Other hyper-parameter values for the RNN models are indicated in Table 5.10. We found that the Adam gradient descent optimisation to be more stable and converge faster than the stochastic gradient descent optimiser.

Table 5.10: Hyper-parameters for RNN models.

Hyper-parameter	Value
Loss function	Logarithm of the hyperbolic cosine
Learning rate	0.001
Mini-batch size	64
Activation function for output layer	Identity function
Dropout rate for dropout layer	0.5
Optimiser	Adam

Training the RNNs took significantly longer because GPU support was not available for the GRU layer for the specific version of the TensorFlow software library we made use of. Training a model for 100 epochs took approximately 16 hours. Using the most up to date version of the software would improve training times significantly. The validation results are given in Table 5.11 for three models with varied output sizes, trained over 100 epochs each.

Table 5.11: Comparison of RNNs with variable output size of the recurrent layer.

GRU output size	MAPE	RMSE	R^2
64	16.4	22.5	0.979
96	13.4	22.5	0.979
128	13.5	22.0	0.980

The model with a GRU layer with 96-element vector output performed best and was trained to convergence over 250 epochs. The resulting model's MAPE, RMSE and R^2 measures are listed in Table 5.12.

Table 5.12: Validation results for best-performing RNN with GRU output size of 96.

Accuracy measure	Result
MAPE	14.1 %
RMSE	22.8 m^3/s
R^2	0.978

The training for the additional 100 epochs reduced the performance of the model for all accuracy measures. When assessing the training and validation loss over the 250 training epochs displayed in Figure 5.13, we can see that the validation loss steadily increases from 20 epochs onward, indicating that the model overfits the data. It seems that additional regularisation, over and above the dropout layer, is needed to prevent overfitting. Applying variation dropout (Gal and Ghahramani, 2016) to the recurrent layer may be a solution to this problem, but was not investigated in this thesis.

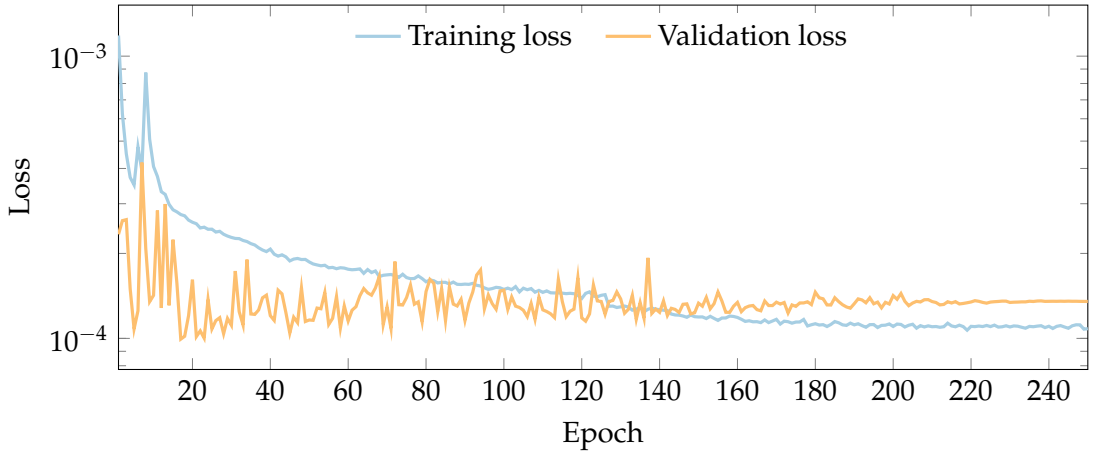


Figure 5.13: Training and validation loss when training RNN with GRU layer with a 96-element vector output to convergence.

Predicted and recorded flows for July 2013 at Marksdrift gauging station are displayed in Figure 5.14. The graph shows well-matched peaks and troughs for high and low flow regimes. In terms of flow magnitude the high flow regime, above 80 m^3/s, displays lower accuracy for this sample of data. This trend is, however, not observed across the validation data set.

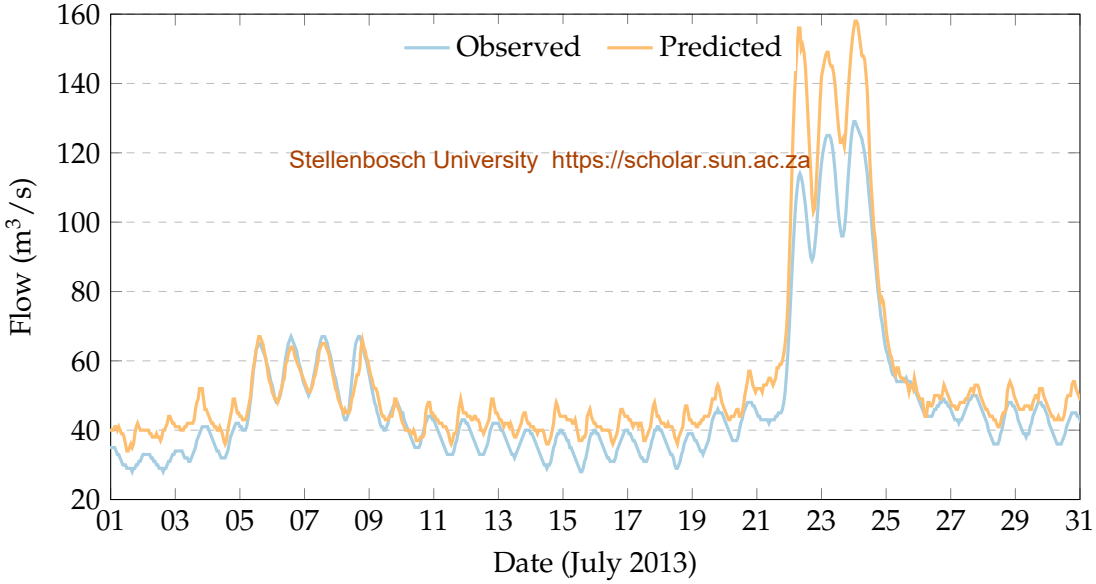


Figure 5.14: Sample flow rate predictions at Marksdrift station for best performing RNN, using GRU layer with 96-element output.

5.8 Summary

We established baseline performance measures using a simple linear regression model. This model performed surprisingly well, with a MAPE of 15 % on the validation set. Plotting the predicted against the observed flows for the validation data set showed that this model does not predict the timing of peaks and troughs well for low flow regimes (below 80 m^3/s). We observe similar performance and disparities between predicted and recorded flow for a single-layer fully-connected neural network. Multiple-layer fully-connected networks showed some improvements on RMSE and R^2 measures as well as significant improvements in the timing of peaks and troughs for lower flow regimes, but no improvement on the MAPE measure. The best performing CNN model displayed the same ability to predict the timing of flow increases and decreases, as well as a marked improvement of the MAPE accuracy (13.6 %). The best performing RNN, although displaying some evidence of overfitting, performed comparative to the best-performing CNN, with a slightly worse MAPE (14.1 %).

The improvement in the prediction of the attenuation of flow peaks and troughs from a linear regression model to the results produced by multi-layer neural networks indicates that the ability of ANNs to model non-linear relationships between input and output is being utilised.

Table 5.13: Comparing performance on the validation set for different network architectures.

Model architecture	MAPE	RMSE	R^2
Linear regression	15.0	23.1	0.978
Single-layer ANN	15.0	23.6	0.977
Multi-layer ANN	15.0	21.3	0.981
CNN	13.6	22.6	0.979
RNN	14.1	22.8	0.978

Test set results

To provide an unbiased view of model performance, we also produce predictions for the test data set. Since this data set has not been used in model selection it did not affect any architecture or hyper-parameter decisions. Results for the test data set is listed in Table 5.14.

Table 5.14: Comparing performance on the test set for different network architectures.

Model architecture	MAPE	RMSE	R^2
Linear regression	16.9	55.4	0.957
Single-layer ANN	16.9	150.0	0.685
Multi-layer ANN	16.6	84.6	0.900
CNN	14.5	49.3	0.966
RNN	15.7	56.5	0.955

The MAPE results of the test set for all models compare favourably with the MAPE results of the validation set. However, the RMSE and R^2 results display significantly lower accuracy. On closer inspection of the test data set, it was found that the large discrepancies between validation and test results were largely caused by a flood event which was part of the test set. Since the model would be used during lower flow regimes (below 200 m^3/s), the lower accuracy during flood-level flows (above 1 000 m^3/s) does not disqualify the model from practical application.

The remainder of the thesis will focus on expanding the accuracy and the utility of the best performing CNN architecture by investigating how to structure neural networks to account for inflows from tributaries, and improving the performance of the model by attempting to account for seasonal variation in abstractions and losses.

Chapter 6

Modelling tributaries

A river's morphology is rarely as simple as the solitary river reach between Vanderkloof Dam and the Marksdrift Weir modelled in the previous chapter. Rivers typically have tributaries that contribute to the flow volume in the modelled river reach. This is also the case for the Orange River, where the Vaal River joins it just downstream of Marksdrift Weir. As we established in Chapter 2, the Vaal River contributes significant volumes of water to the Orange River's flow, but these contributions are often sporadic and unpredictable. Anticipating the contributions would allow reduced releases from the Vanderkloof Dam which would amount to significant savings in terms of water resources.

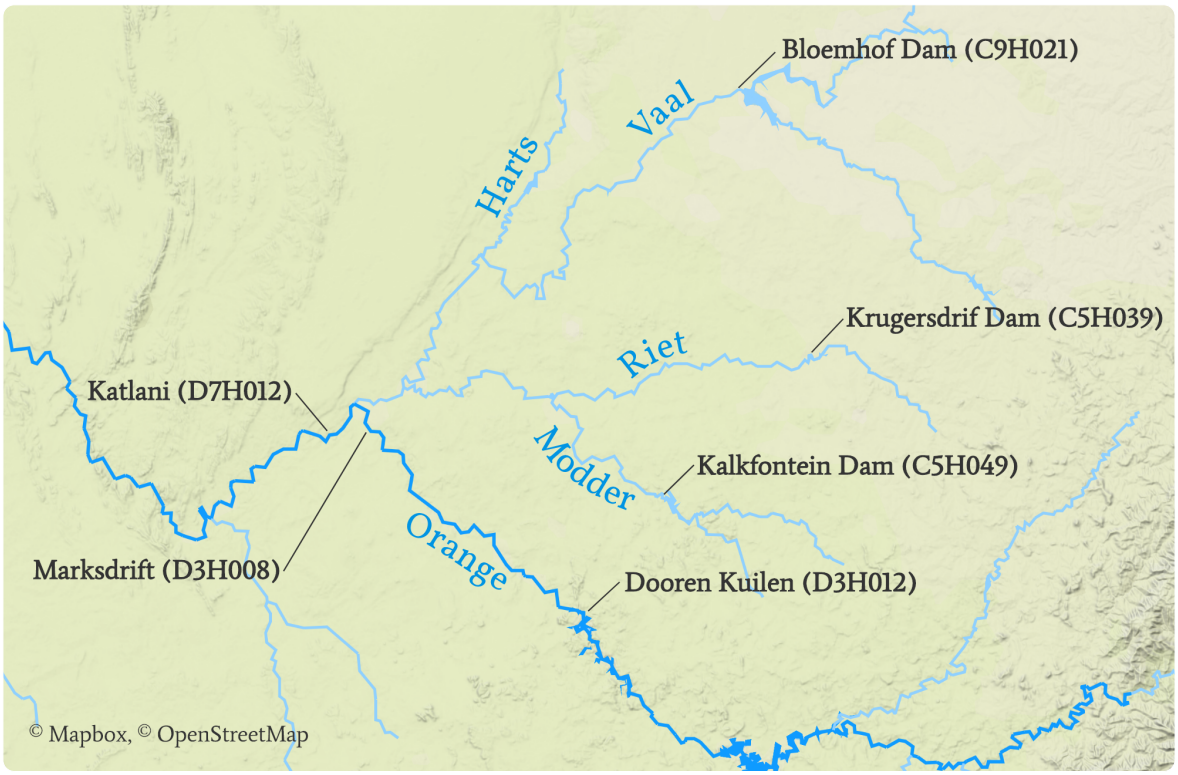


Figure 6.1: Flow gauging stations used for modelling the combined flow from the Orange and Vaal Rivers recorded at Katlani (D7H012).

The Vaal River joins the Orange River 200 km downstream of the Vanderkloof Dam. If one considers the Orange and Vaal River systems as a single water resource that satisfies downstream requirements, it would allow the volume of water from the Vaal River to be taken into account when planning releases from the Vanderkloof Dam.

As displayed in Figure 6.1, two of the Vaal River's tributaries, the Riet and Harts Rivers, join it approximately 35 km and 155 km upstream of the Orange-Vaal confluence, respectively. The Riet River is joined by the Modder River 125 km upstream of its confluence with the Vaal River. In order to take into account contributions from these tributaries, we would need to anticipate the inflows for the same input time window we use for the releases from the Vanderkloof Dam, a period of four days. At the Vanderkloof Dam, we know what the release schedule will be ahead of time, but along the Vaal River system this would not be the case if we use recorded flows at structures, such as weirs, which do not control their outflow. For this reason we propose using outflows at three dams, which do control their releases, the Bloemhof Dam on the Vaal River, the Krugersdrift Dam on the Modder River, and the Kalkfontein Dam on the Riet River. Unfortunately there is no data available for the Spitskop Dam on the Harts River, so the inflows from this tributary are not taken into account.

We need to adjust the model input parameters and architecture from Chapter 5 to allow us to add these additional input values. We will attempt to predict the flow rates at the Katlani (D7H012) gauging station, just downstream of the Orange-Vaal confluence, by including contributions from the Vaal River tributaries using two different approaches:

- including inflows from tributaries in a multi-dimensional input data set, and
- using a parallel model architecture mirroring the physical river structure.

6.1 Establishing a baseline

We set up a baseline model using the CNN10 architecture (refer to Figure 5.10), by only using input from the flow rates recorded at the Dooren Kuilen (D3H012) station at Vanderkloof Dam, and train the model to predict flow rates at Katlani station. We include no inflows from tributaries on the Vaal River system at this point. Figure 6.2 plots the training and validation loss over 500 training epochs.

From Figure 6.2 we can see that the model severely overfits the training data when not given inputs from tributaries. This means that the model resorts to memorising the training data and that it would not generalise well to the validation data set. The validation results in Table 6.1 confirms that the model does not produce accurate results.

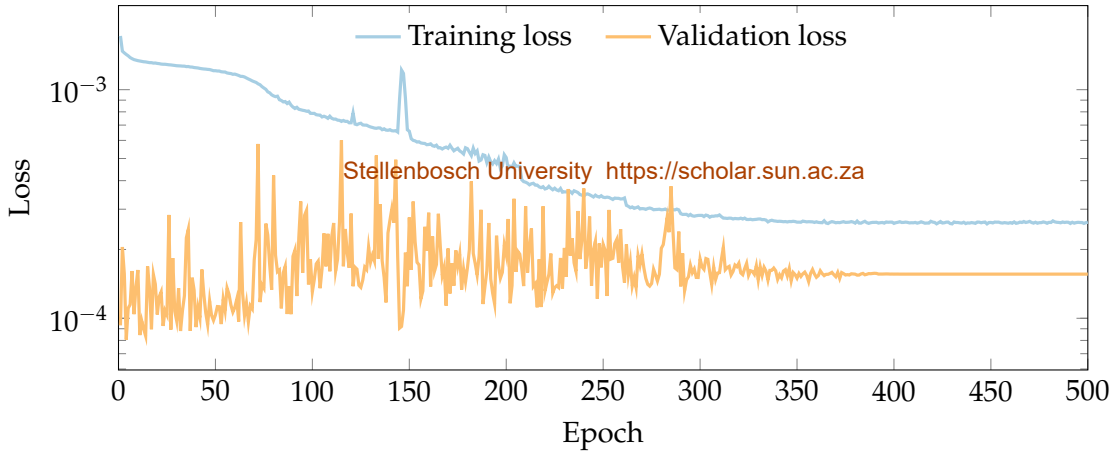


Figure 6.2: Training and validation loss when training baseline CNN10 model to convergence, excluding inflows from tributaries. Rising validation loss indicates overfitting.

Table 6.1: Validation results at Katlani station for CNN10 model, excluding inflows from tributaries.

Accuracy measure	Result
MAPE	38.6 %
RMSE	61.9 m^3/s
R^2	-0.649

Considering a sample prediction for February to March 2017, displayed in Figure 6.3, we can see that the flow from the Vaal River system is not anticipated by the model. We specifically chose the February to March 2017 period for comparison and analysis because the Vaal River system contributes a large portion of the flow in the Orange River during this time.

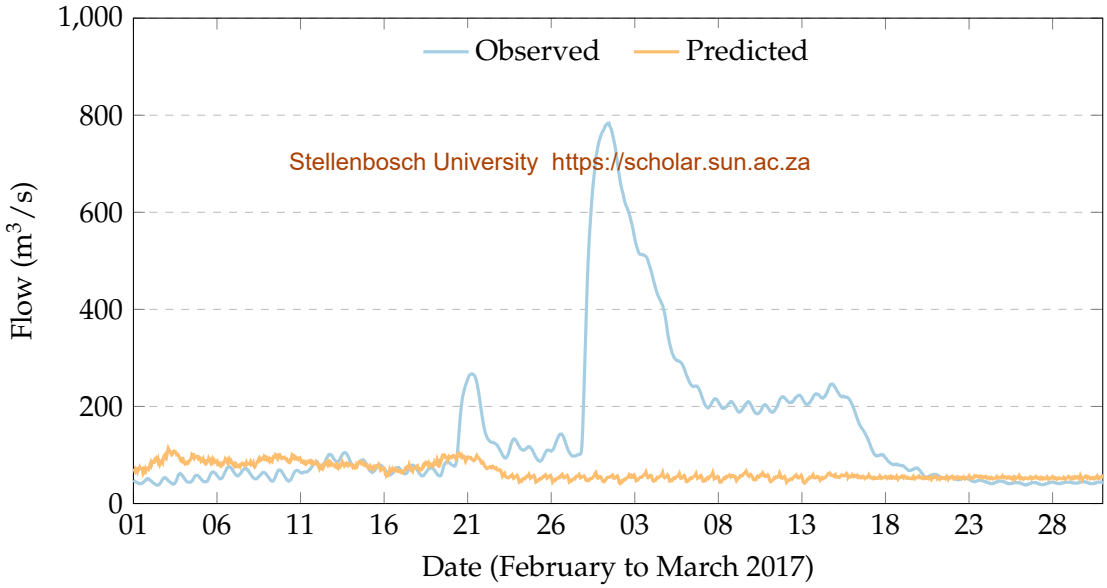


Figure 6.3: Sample of flow rate predictions at Katlani gauging station for the baseline CNN10 model, excluding inflows from tributaries.

6.2 Including inflows from tributaries in multi-dimensional input

For our first experiment we propose layering the inflow values from the tributaries and the main river reach on top of one another in a matrix. By adding another dimension to our input vector $\mathbf{x}$, making it a matrix $\mathbf{X}$, we include inflows from the tributaries along this additional dimension. Our training samples now take the form:

$$\begin{bmatrix} x_i^{(o)} & x_{i+1}^{(o)} & x_{i+2}^{(o)} & \cdots & x_{i+n}^{(o)} \\ x_i^{(v)} & x_{i+1}^{(v)} & x_{i+2}^{(v)} & \cdots & x_{i+n}^{(v)} \\ x_i^{(m)} & x_{i+1}^{(m)} & x_{i+2}^{(m)} & \cdots & x_{i+n}^{(m)} \\ x_i^{(r)} & x_{i+1}^{(r)} & x_{i+2}^{(r)} & \cdots & x_{i+n}^{(r)} \end{bmatrix} \mapsto y_{i+n}, \tag{6.1}$$

with $x_i^{(o)}$, $x_i^{(v)}$, $x_i^{(m)}$ and $x_i^{(r)}$ representing the flow rate at time step i on the Orange, Vaal, Modder and Riet Rivers, respectively. The flow rate at time step $i+n$ at the Katlani gauging station is represented by y_{i+n}.

The model architecture we use is identical to the CNN10 model architecture, discussed in Chapter 5, other than the input vector $\mathbf{x}$ changing to an input matrix $\mathbf{X}$. We will refer to this model as CNN10-MATRIX. Performance over 500 training epochs for the training and validation loss is displayed in Figure 6.4.

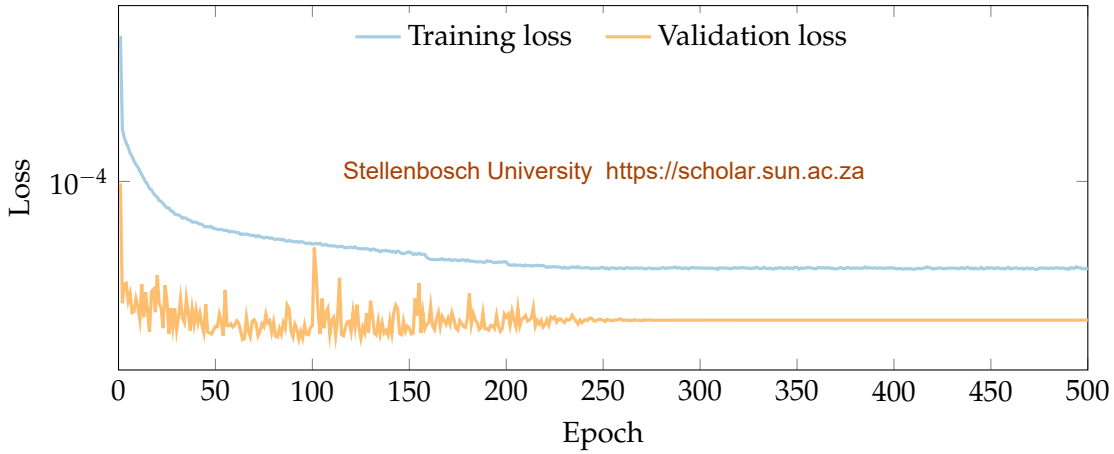


Figure 6.4: Training and validation loss for CNN10-MATRIX.

A sample of predictions for February to March 2017 are displayed in Figure 6.5 and validation set results are given in Table 6.2. From the sample prediction we can see that the model predicts the timing of the increase in flow rate on 28 February, and the drop in flow rate on 18 February, relatively well. The smaller peak in flow rate around 21 February is not predicted at all. On closer inspection of the input data, we confirmed that none of the stations included in this model registered an increase in flow rate preceding that time. This suggests that this peak in flow rate originated from a different tributary which is not considered in the model, possible the Harts River.

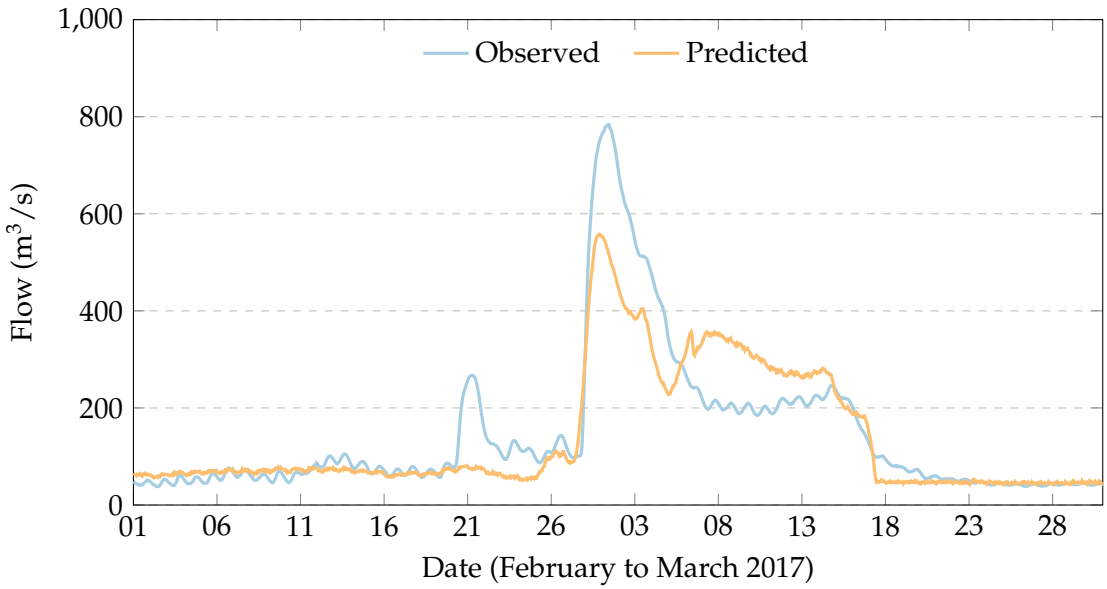


Figure 6.5: Sample of flow rate predictions at Katlani gauging station for the CNN10-MATRIX model.

Table 6.2: Validation results at Katlani station for CNN10-MATRIX.

Accuracy measure	Result
MAPE	20.6 %
RMSE	23.1 m^3/s
R^2	0.774

6.3 Parallel model architecture

For our second experiment we take inspiration from the physical nature of the tributaries and use a parallel model structure as displayed in Figure 6.6. This architecture attempts to give contributions from each river the opportunity to be simulated separately before being combined in a 2048-node fully-connected layer. The structure of each arm of the network is identical to the CNN10 architecture.

Figure 6.6: CNN10-PARALLEL model, where four CNN10 models feed into a fully-connected layer.

For our training data, each of the rows in the input matrix defined in Equation 6.1 will be used as input for each of the parallel networks. Training the model to convergence over 500 epochs yields the training and validation loss trends displayed in Figure 6.7. The slight upward trend of the validation loss indicates that the model has started overfitting the training data from approximately epoch 200.

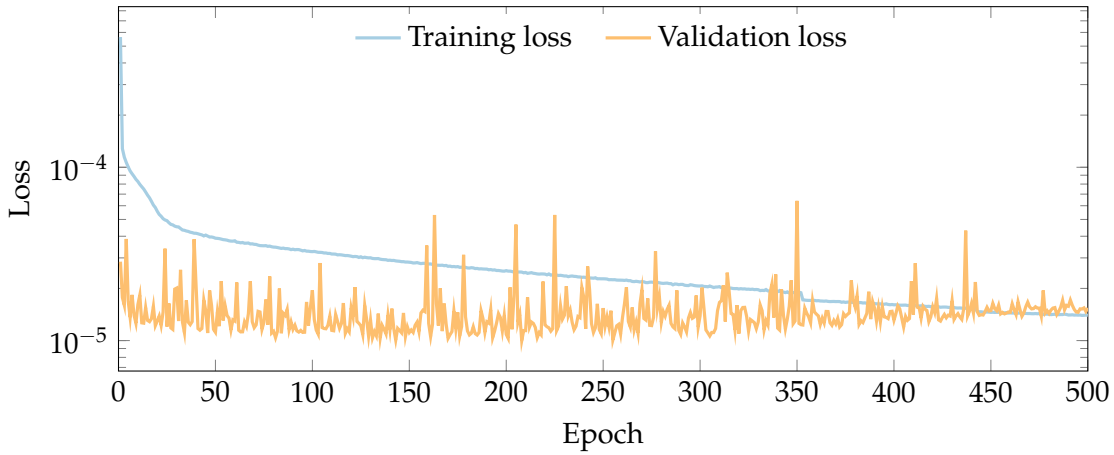


Figure 6.7: Training and validation loss when training CNN10-PARALLEL model to convergence.

Validation results for the CNN10-PARALLEL model are listed in Table 6.3. The results achieved display slightly lower accuracy across all measures than the CNN10-MATRIX model results. Assessing the predicted flow rate at the Katlani gauging station in Figure 6.8 reveals that the model predicts the timing of the increase in flow rate, and subsequent decrease, well. As expected, the smaller peak in flow rate on the 21st of February is not predicted by the model.

Table 6.3: Validation results at Katlani station for CNN10-PARALLEL.

Accuracy measure	Result
MAPE	23.2 %
RMSE	26.0 m^3/s
R^2	0.713

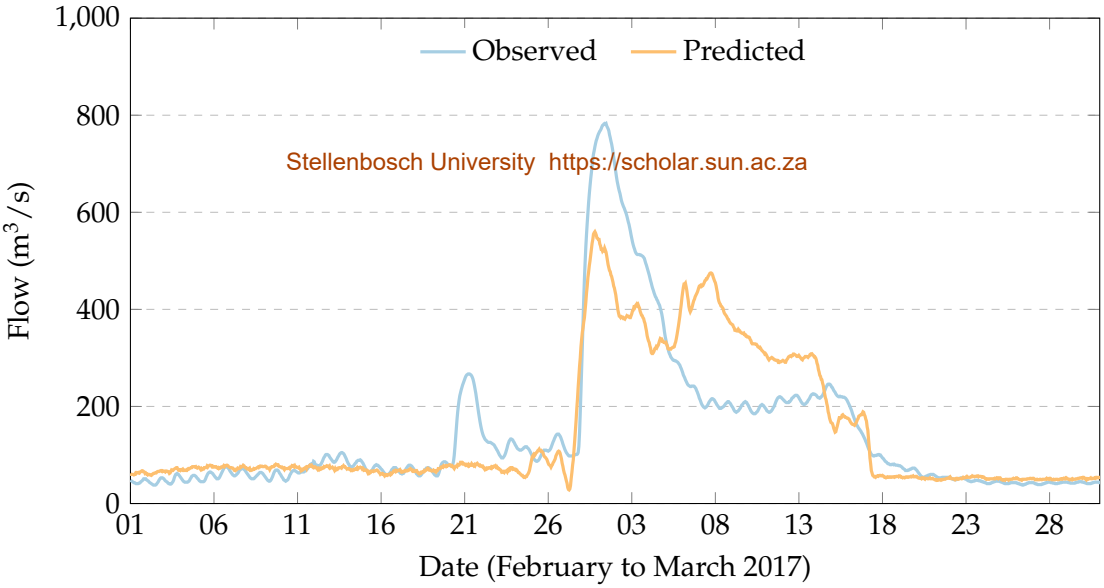


Figure 6.8: Sample of flow rate predictions at Katlani gauging station for the CNN10-PARALLEL model.

6.4 Test set results

The test set results listed in Table 6.4 show similar results for MAPE, but reduced accuracy for RMSE and R^2 results. A flood event forming part of the test set was the cause of this discrepancy, with large errors in predictions for this time period contributing a disproportionate amount to these values.

Table 6.4: Test result comparison.

Model	MAPE	RMSE	R^2
CNN10-MATRIX	21.6	52.0	0.948
CNN10-PARALLEL	23.5	81.5	0.871

6.5 Summary

Including inflows from the Vaal River tributary is essential when optimising releases from the Vanderkloof Dam. The inaccurate predictions produced by our baseline model, which included no data from the Vaal River tributary, indicated that including the additional data is required to produce a model that predicts flow rates well at Katlani, and other downstream stations. We proposed two different techniques to include this additional input, both effectively predicting the timing of increases and decreases in flow rate in the main river reach, although lacking accuracy in term of

the magnitude of the contributions. Further model refinement, such as employing an early-stopping technique once overfitting is detected, or additional regularisation, may result in increased model performance.

Chapter 7

Modelling seasonal factors

Losses and abstractions from the Orange River display an annual cycle, with lower values during the winter season, and higher values in the summer season (refer to Figure 2.6). We do not provide any input data that could be used to deduce the effects of this cycle to the models set up in Chapter 5, and they could therefore not take this seasonal variation into account. In this chapter we investigate the inclusion of additional seasonal data as input to the models, i.e.

- including the time of the year in the input,
- including evaporation data in the input, and
- including errors made by the model in the recent past as input for subsequent predictions.

7.1 Establishing a baseline

We use the CNN10 model trained for predicting flow rates at Marksdrift station in Chapter 5 as a baseline model, and compare the models we set up in this chapter against it. Since we are using evaporation data from a meteorological station, with a data set beginning in 1990, as input to one of the models set up in this chapter, we update the validation date range to match this reduced training data range. This adjustment is necessary to compare models on an equal basis, since validation results should be calculated from the same data set. The adjusted baseline validation results for the CNN10 model is given in Table 7.1.

Table 7.1: Validation results at Marksdrift station for CNN10 baseline model.

Accuracy measure	Result
MAPE	13.6 %
RMSE	15.6 m^3/s
R^2	0.895

Figure 7.1 displays a sample of predictions from the validation set, for the first week of August 2013. We hypothesise that the differences between the observed and predicted flows for this time period are typical of errors caused by seasonal variations, since they are relatively constant over a relatively long period of time. We suggest that this constant nature of the errors mirrors the locally constant nature of the seasonal factors that cause them. For instance, cooler weather conditions causing lower than normal evaporation losses would typically persist for a number of days.

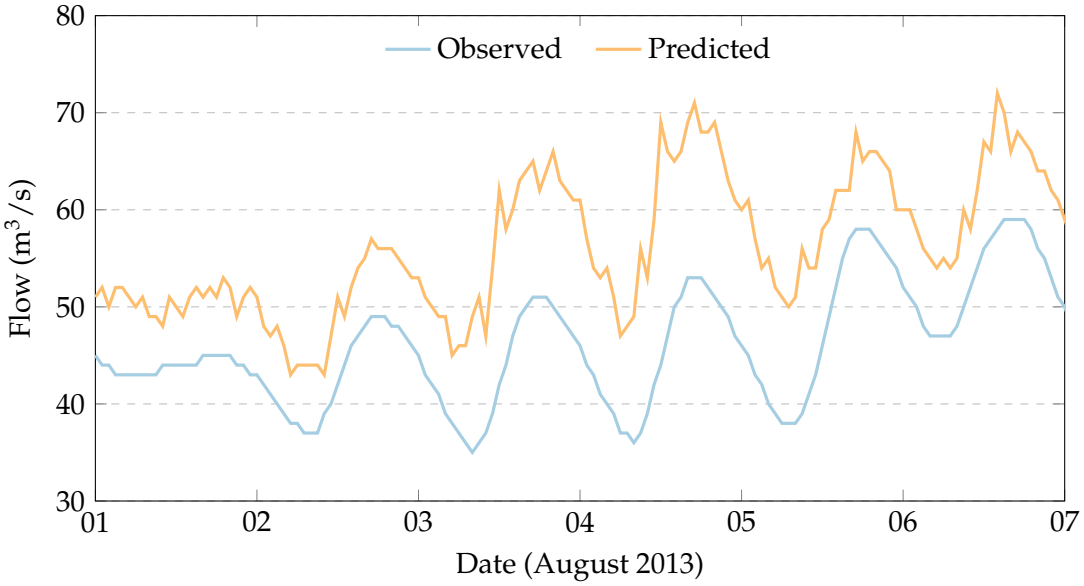


Figure 7.1: Sample of flow rate predictions at Marksdrift for the CNN10 baseline model showing errors possibly caused by seasonal factors.

7.2 Including time of year in input

Including the time of the year as part of the input data is a natural and simple solution to giving the model an indication of the season of the year. We construct our training samples by adding two values, the sine and cosine of the normalised day of the year. This effectively gives a unique combination for each day of the year, and would prevent the model confusing two distinct days. As observed with the seasonal trend of losses and abstraction, these functions are continuous across different years, with no discontinuity between 31 December and 1 January. Figure 7.2 shows the input values for each day of the year.

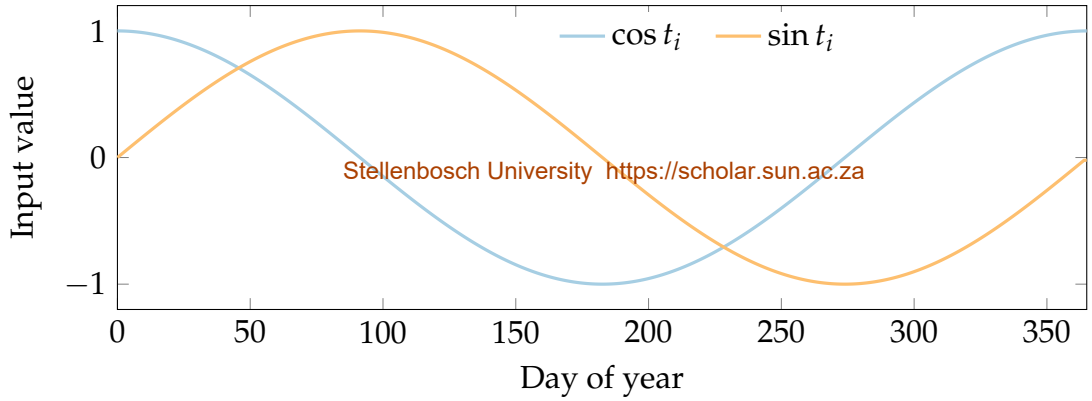


Figure 7.2: Time values included as input into CNN10-TIME model.

We now combine these values with our flow data to produce training samples in the format:

$$\begin{bmatrix} x_i & x_{i+1} & x_{i+2} & \cdots & x_{i+n} \\ \sin t_i & \sin t_{i+1} & \sin t_{i+2} & \cdots & \sin t_{i+n} \\ \cos t_i & \cos t_{i+1} & \cos t_{i+2} & \cdots & \cos t_{i+n} \end{bmatrix} \mapsto y_{i+n}, \tag{7.1}$$

with:

$$t_i = 2\pi \frac{\text{day of year}}{365.25}. \tag{7.2}$$

x_i and y_{i+n} represent the flow rates at the upstream and downstream stations, respectively.

We use the CNN10 architecture from Chapter 5, with a matrix $\mathbf{X}$ as input layer to accommodate the additional data. We increase the batch size from 64 to 168 to prevent overfitting. Training and validation loss over 100 training epochs are displayed in Figure 7.3 for the CNN10-TIME model.

Validation results and a sample of predictions are given in Table 7.2 and Figure 7.4, respectively. The trained model shows a slight improvement over all accuracy measures, but no apparent reduction in the constant differences between predicted and observed flow rates in the sample of data in Figure 7.4.

7.3 Including evaporation in input

Including recorded evaporation data as input data may give our model the required context to predict the seasonal influence of losses and abstractions. As indicated in

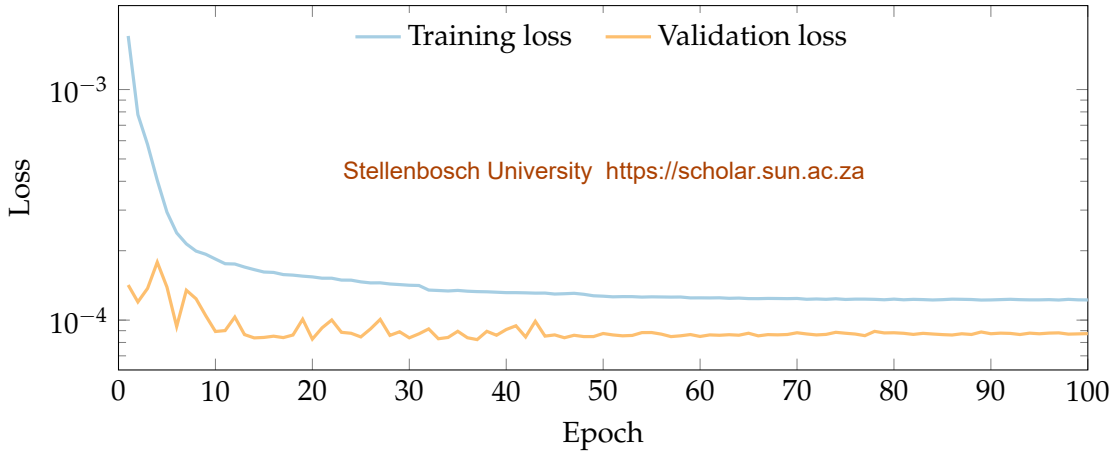


Figure 7.3: Training and validation loss when training CNN10-TIME model.

Table 7.2: Validation results at Marksdrift station for CNN10-TIME.

Accuracy measure	Result
MAPE	13.1 %
RMSE	13.9 m^3/s
R^2	0.917

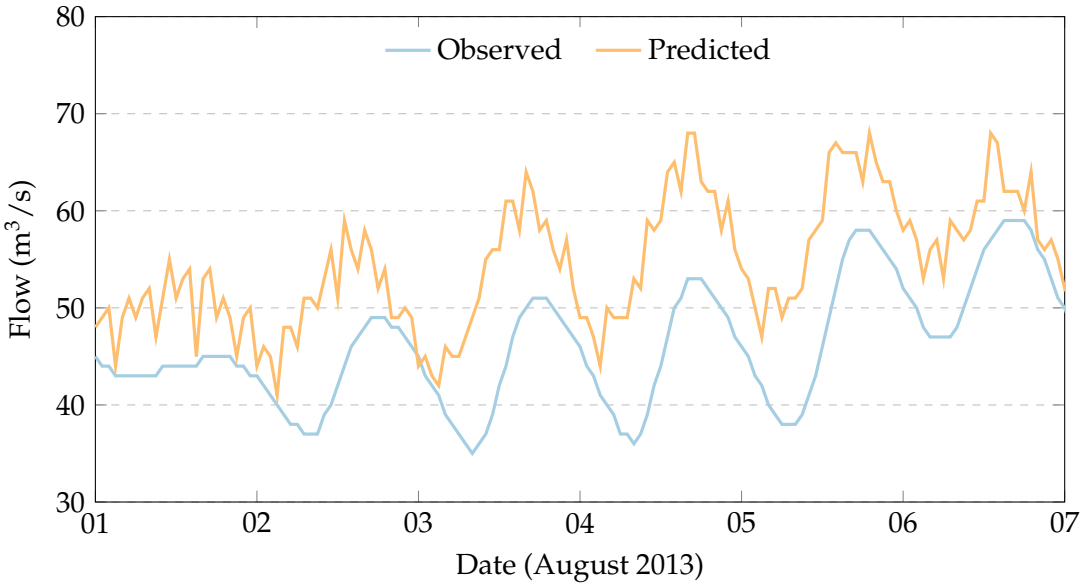


Figure 7.4: Sample of flow rate predictions at Marksdrift for CNN10-TIME.

Section 4.4, in an operational setting we would only have evaporation data available up to the point in time we run a prediction. This is unlike flow rate, which we would have access to from the Vanderkloof Dam's release schedule. The limitation on data availability means we have to offset our evaporation data to prevent the model from

"seeing into the future". Our training samples will take the form:

$$\begin{bmatrix} x_i & x_{i+1} & x_{i+2} & \cdots & x_{i+n} \\ e_{i-n} & e_{i-n+1} & e_{i-n+2} & \cdots & e_i \end{bmatrix} \mapsto y_{i+n}, \tag{7.3}$$

with **x** and **e** representing flow and evaporation input vectors, respectively. Note that the time-steps for evaporation data starts n time-steps before the first time-step for input flow data.

We make use of the CNN10 architecture, with matrix input, and an increased batch size of 168 to prevent overfitting. Input for evaporation data is sourced from the Vanderkloof (D3E003) meteorological station. Training the CNN10-EVAP model to convergence over 100 epochs yields the training and validation losses displayed in Figure 7.5.

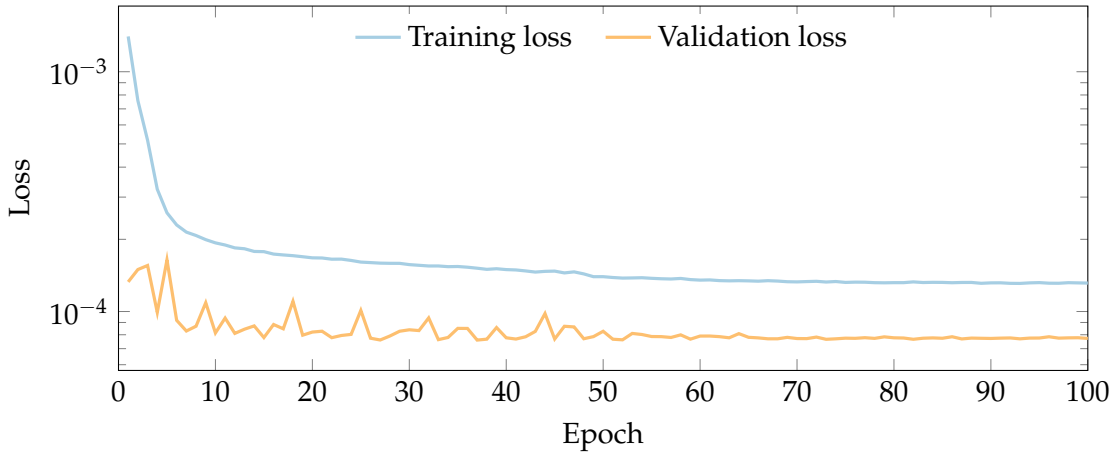


Figure 7.5: Training and validation loss when training CNN10-EVAP model to convergence.

Validation results for the CNN10-EVAP model are listed in Table 7.3. The results show some improvement across all measures indicating that the model may have deduced a relationship between historic evaporation values and seasonal losses from the river. The increased accuracy is, however, not significant enough to make a visible difference to the sample of predictions displayed in Figure 7.6.

7.4 Including recent errors in input

If we consider the errors in the baseline CNN10 model, displayed in Figure 7.1, we observe that the difference between the recorded and predicted flow rates seem to stay more-or-less constant over time. This indicates that the errors are caused by factors

Table 7.3: Validation set results at Marksdrift station for CNN10-EVAP model.

Accuracy measure	Result
MAPE	13.0 %
RMSE	12.9 m^3/s
R^2	0.928

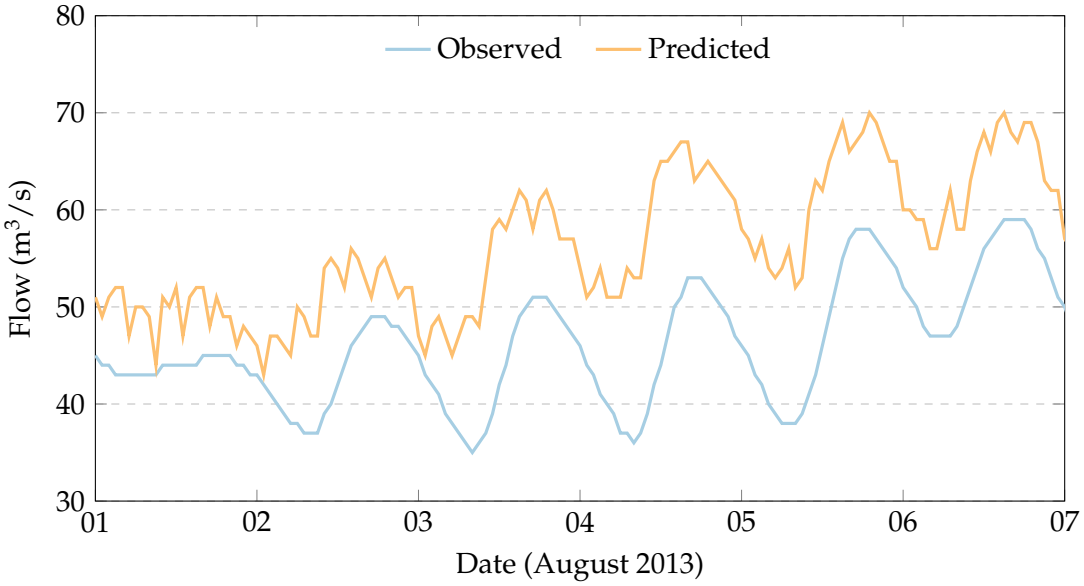


Figure 7.6: Sample of flow rate predictions at Marksdrift for CNN10-EVAP.

that consistently cause higher or lower flow rates in the river. Because of the seemingly constant nature of the errors over short time periods, we can leverage this additional information and feed the errors made by the CNN10 model in the recent past (which would be known in an operational setting) as input into a new model.

We propose including both the recorded and predicted values during the preceding time-steps, instead of just the difference between them. We reason that including both the sets of data would give the model additional information about the nature of the error. Observations of errors in the validation set indicate that errors that occur at high flow regimes are often higher than the errors immediately preceding it in lower flow regimes. For example, in Figure 5.12, errors preceding the increased flow rate on 22 July are lower than the errors subsequent to the increase. We hypothesise that a neural network may be able to make the distinction between errors during high and low flow regimes when furnished with both sets of data.

The training samples for our new model will be in the form:

$$\begin{bmatrix} x_i & x_{i+1} & x_{i+2} & \cdots & x_{i+n} \\ p_{i-n} & p_{i-n+1} & p_{i-n+2} & \cdots & p_i \\ o_{i-n} & o_{i-n+1} & o_{i-n+2} & \cdots & o_i \end{bmatrix} \mapsto y_{i+n}. \tag{7.4}$$

Here **x** represents the flow rate at the upstream station and **p** and **o** are the predicted and observed flow values at the downstream station for the n time-steps preceding prediction time i. As with evaporation, we take care to not allow the model access to data that would not be available in an operation setting and at prediction time. Training the CNN10-DIFF model to convergence over 100 epochs gives training and validation losses as shown in Figure 7.7.

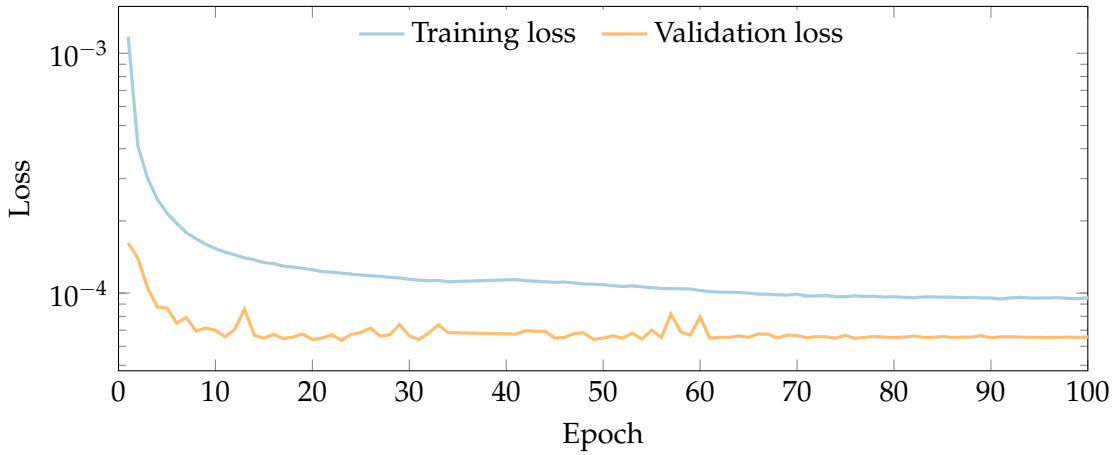


Figure 7.7: Training and validation loss when training CNN10-DIFF model to convergence.

Table 7.4 lists validation results which display a marked increase in performance across all measures. Figure 7.8 also displays a clear improvement of flow rate accuracy for the sample time period. The results indicate that this approach effectively offsets the constant error between predicted and observed flow rates.

Table 7.4: Validation set results at Marksdrift station for CNN10-DIFF model.

Accuracy measure	Result
MAPE	10.6 %
RMSE	12.0 m^3/s
R^2	0.938

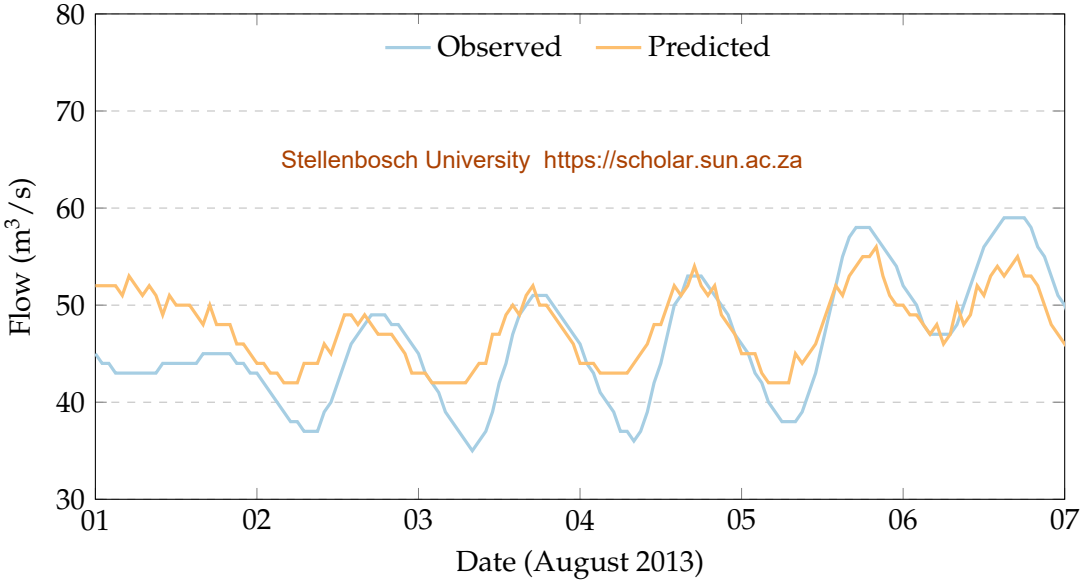


Figure 7.8: Sample of flow rate predictions at Marksdrift for CNN10-DIFF.

7.5 Summary

We attempt to account for seasonal variation in flows by including the time of the year, evaporation data and the recent differences between predicted and observed flow rates as input into the models. These approaches displayed small to significant levels of success, with the comparative accuracy measures given in Table 7.5.

Table 7.5: Comparing validation results produced by including different seasonal data.

Model	Additional data	MAPE	RMSE	R^2
CNN10	None	13.6	15.6	0.895
CNN10-TIME	Time of year	13.1	13.9	0.917
CNN10-EVAP	Evaporation	13.0	12.9	0.928
CNN10-DIFF	Recent errors	10.6	12.0	0.938

Although including time of year and evaporation showed some improvement in model performance, including recent history of observed and predicted flows was the most effective at reducing errors caused by seasonal factors.

Test set results

Predictions against the test set, listed in Table 7.6, shows comparative results for the MAPE accuracy, but reduced accuracy for RMSE and R^2. As discovered in Chapter 5, a flood event forming part of the test set is largely the cause of this discrepancy. It is encouraging that the MAPE for our best-performing CNN10-DIFF model is not

affected by the flood event, indicating that it succeeds at predicting relatively accurate flow rates even at flood-level flows.

Table 7.6: Comparing test results produced by including different seasonal data.

Model	Additional data	MAPE	RMSE	R^2
CNN10	None	18.4	57.2	0.969
CNN10-TIME	Time of year	14.4	89.9	0.923
CNN10-EVAP	Evaporation	14.8	103.7	0.898
CNN10-DIFF	Recent errors	10.2	42.9	0.982

Chapter 8

Conclusions and future work

We set out to develop an accurate and robust flow routing model of the Orange River downstream of the Vanderkloof Dam using deep learning techniques. Such a model would allow the operators of the dam, the DWS, to optimise releases by taking into account inflows from the Vaal River system and factors that cause seasonal variations in flow rates.

Deep learning models are particularly suited to problems where the relationships between the different factors that influence the outcome are not well defined. River flow routing over long distances can be seen as such a problem, since there are numerous factors that influence the flow volume and rate, as discussed in Chapter 2. We hypothesised that a machine learning model would "learn" how these factors affect downstream flow rate by being trained on historical data. In Chapter 4 we confirmed that the data collected by the DWS is suitable in terms of quantity and quality to train deep learning models on.

We discussed the theory underlying three neural network model architectures which we investigated in Chapter 3: fully-connected neural networks, convolutional neural networks (CNNs) and recurrent neural networks (RNNs). In Chapter 5 we trained a variety of these models and established that multi-layer fully-connected networks, CNNs and RNNs outperform a simple linear regression model and have the ability to predict the timing of peaks and troughs in flow rate over a distance of 174 km, from the Vanderkloof Dam to the gauging station at Marksdrift.

In Chapter 6 we expanded on how to include contributions from tributaries. To account for inflows from tributaries we developed a parallel neural network model structure, as well as a CNN model which makes use of multi-dimensional input of flow rates from tributaries. Both these approaches allowed us to predict increased flow rates at a station downstream of the Vaal River confluence when there is an increase in flow rate along the tributary. This suggests that the models have the ability to account for inflows from tributaries and, with some further refinement, could be used as a tool to optimise releases from the Vanderkloof Dam.

We attempted to account for factors that cause seasonal variation in the flow rate in Chapter 7 by including additional data in the input data that may provide seasonal context. Including the time of the year or evaporation data showed slight improvements in model performance, while including recent errors as input in subsequent model runs showed dramatic improvement over all accuracy measures (reducing mean absolute percentage error (MAPE) from 13.6 % to 10.6 %). We hypothesise that this increased accuracy is because factors that cause seasonal variation in flow rates do not change rapidly, and therefore errors made in the recent past are a good predictor for errors that would be made in the future.

8.1 Future work

Before undertaking future work in this problem domain, one should note that other statistical techniques, such as support vector regression models (Steyn, 2018), have shown promise in the area of river flow routing, and these approaches should also be considered before deciding on the practical implementation of neural networks.

While we believe we achieved the goal of proving that a data-driven approach to river flow routing is viable along the Orange River using artificial neural network models, we did not exhaust all opportunities to improve our models' performance and utility. Some refinements which are considered promising are:

- to improve model performance by alternative regularisation techniques for models that display signs of overfitting,
- to develop a stable purpose-built loss function that is in line with the MAPE measure of accuracy,
- to investigate cross-validation techniques (such as k-fold cross-validation) to obtain a better view on how well the models generalise,
- to combine the CNN10-MATRIX model architecture with the additional data inputs used in the CNN10-DIFF model,
- to combine evaporation, time-of-the year and recent errors as inputs into a single model,
- to investigate deeper model architectures, such as ResNet (He et al., 2015a) and U-Net (Ronneberger et al., 2015), and
- to produce probability estimates for predictions (Gal, 2016), providing us with a metric to measure how certain an ANN model is of its prediction.

Considering the comparatively low cost of setting up a purely data-driven machine learning solution, and the simplicity of the building blocks which were developed to solve the flow routing problem, we believe that this approach could be practically implementable in a decision support system to assist the operators of the Vanderkloof Dam.

Bibliography

American Society of Civil Engineers. (2000a). Artificial neural networks in hydrology. i: Preliminary concepts. *Journal of Hydrologic Engineering*, *5*(2), 115–123. doi:10.1061/(ASCE)1084-0699(2000)5:2(115)

American Society of Civil Engineers. (2000b). Artificial neural networks in hydrology. ii: Hydrologic applications. *Journal of Hydrologic Engineering*, *5*(2), 124–137. doi:10.1061/(ASCE)1084-0699(2000)5:2(124)

Bai, S., Kolter, J. Z. & Koltun, V. (2018). An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. arXiv: 1803.01271. Retrieved from http://arxiv.org/abs/1803.01271

Briers, C. J. (2018). Thesis code: Data driven river flow routing using deep learning: Predicting flow along the lower orange river, southern africa. GitHub. Retrieved from https://github.com/CJBriers/orange-river-thesis

Cho, K., Van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1724–1734). doi:10.3115/v1/D14-1179

Chung, J., Gulcehre, C., Cho, K. & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

De Saint-Venant, A. J. C. B. (1871). Theorie du mouvement non permanent des eaux, avec application aux crues des rivieres et a l'introduction de marees dans leurs lits. *Comptes rendus des seances de l'Academie des Sciences*, *36*, 174–154.

Department of Water and Sanitation. (n.d.[a]). Boegoeberg Dam. Retrieved November 30, 2018, from http://www.dwaf.gov.za/Orange/Low_Orange/boegoebe.aspx

Department of Water and Sanitation. (n.d.[b]). Hydrological Services, Surface Water. Retrieved November 30, 2018, from http://www.dwa.gov.za/Hydrology/Verified/hymain.aspx

Fair, K. (2003). *Operational Model of the Orange River* (tech. rep. No. 865-1-03). Water Research Commission.

Fei-Fei, L. & Karpathy, A. (2015). CS231n, Convolutional Neural Networks for Visual Recognition. Retrieved November 30, 2018, from http://cs231n.github.io/neural-networks-2/#init

Fick, S. E. & Hijmans, R. J. (2017). WorldClim 2: New 1-km spatial resolution climate surfaces for global land areas. *International Journal of Climatology*, *37*, 4302–4315. doi:10.1002/joc.5086

Gal, Y. (2016). *Uncertainty in Deep Learning* (Doctoral dissertation, University of Cambridge). Retrieved from http://www.cs.ox.ac.uk/people/yarin.gal/website/thesis/thesis.pdf

Gal, Y. & Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 29* (pp. 1019–1027). Retrieved from http://papers.nips.cc/paper/6241-a-theoretically-grounded-application-of-dropout-in-recurrent-neural-networks.pdf

Gers, F., Schmidhuber, J. & Cummins, F. (1999). Learning to forget: continual prediction with LSTM. In *1999 Ninth International Conference on Artificial Neural Networks* (Vol. 2, pp. 850–855). doi:10.1049/cp:19991218

Glorot, X. & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Vol. 9, pp. 249–256). Retrieved from http://proceedings.mlr.press/v9/glorot10a.html

He, K., Zhang, X., Ren, S. & Sun, J. (2015a). Deep residual learning for image recognition. *CoRR*, *abs/1512.03385*. arXiv: 1512.03385. Retrieved from http://arxiv.org/abs/1512.03385

He, K., Zhang, X., Ren, S. & Sun, J. (2015b). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision* (pp. 1026–1034). doi:10.1109/ICCV.2015.123

Hochreiter, S., Bengio, Y., Frasconi, P. & Schmidhuber, J. (2001). Gradient flow in recurrent nets: The difficulty of learning long-term dependencies. In S. C. Kremer & J. F. Kolen (Eds.), *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Ioffe, S. & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning* (pp. 448–456). Retrieved from http://dl.acm.org/citation.cfm?id=3045118.3045167

Kalbus, E., Reinstorf, F. & Schirmer, M. (2006). Measuring methods for groundwater – surface water interactions: A review. *Hydrology and Earth System Sciences*, *10*(6), 873–887. doi:10.5194/hess-10-873-2006

Kingma, D. P. & Ba, J. (2014). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*. Retrieved from http://arxiv.org/abs/1412.6980

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (pp. 1097–1105).

Le Grange, A. d. P., Briers, C. J. & Hallowes, J. (2009). *Orange River System: Real-Time Operating System for the Lower Orange River Main Study Report* (tech. rep. No. P WMA 14/000/00/0507). Department Water Affairs and Forestry.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541–551. doi:10.1162/neco.1989.1.4.541

Lenail, A. (2018). NN-SVG, Publication-ready NN-architecture schematics. Retrieved November 30, 2018, from http://alexlenail.me/NN-SVG/

Lesotho Highlands Development Authority. (n.d.). Lesotho Highlands Water Project Phase II, Economic Components. Retrieved November 30, 2018, from http://www.lhda.org.ls/phase2/economicimpactcomponents.php

Maré, H. G. & Sejamoholo, B. (2010). *Orange River System Annual Operating Analysis – System Analysis* (tech. rep. No. P RSA C000/00/12210). Department Water Affairs and Forestry.

Minsky, M. & Papert, S. A. (1969). *Perceptrons: An introduction to computational geometry*. Massachusetts Institute of Technology.

Olah, C. (2015). Understanding LSTM Networks. Retrieved November 30, 2018, from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Ramsar. (n.d.). Ramsar sites information service, Orange River Mouth. Retrieved November 30, 2018, from https://rsis.ramsar.org/ris/526

Ronneberger, O., Fischer, P. & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *CoRR*, *abs/1505.04597*. arXiv: 1505.04597. Retrieved from http://arxiv.org/abs/1505.04597

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, *115*(3), 211–252. doi:10.1007/s11263-015-0816-y

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, *3*(3), 210–229. doi:10.1147/rd.33.0210

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D. & Graepel, T. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Simonyan, K. & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, *abs/1409.1556*.

Springenberg, J., Dosovitskiy, A., Brox, T. & Riedmiller, M. (2015). Striving for Simplicity: The All Convolutional Net. In *International Conference on Learning Representations (workshop track)*. Retrieved from http://lmb.informatik.uni-freiburg.de/Publications/2015/DB15a

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*, 1929–1958. Retrieved from http://jmlr.org/papers/v15/srivastava14a.html

Steyn, M. (2018). *Short-term stream flow forecasting and downstream gap infilling using machine learning techniques* (Master's thesis, Stellenbocsh University). Retrieved from http://hdl.handle.net/10019.1/103394

Sutskever, I., Martens, J., Dahl, G. & Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning* (pp. 1139–1147).

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences* (Doctoral dissertation, Harvard University).

Xiong, W., Wu, L., Alleva, F., Droppo, J., X., H. & Stolcke, A. (2018). The Microsoft 2017 conversational speech recognition system. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 5934–5938.

Zhu, X. (2006). Semi-supervised learning literature survey. *Computer Science, University of Wisconsin-Madison*, *2*(3), 4.

Appendix A

Raw data samples

Sections A.1 to A.4 contain four sample files showing the format of data downloaded from the DWS Hydrology website (Department of Water and Sanitation, n.d.[b]).

A.1 Flow gauging data sample

```
Data are continuously updated and reviewed.
The format of this file is as follows:
POS.  1-8   = Date of measurement CCYYMMDD
POS. 10-15  = Time of measurement HHMMSS
POS. 27-35  = Corrected level in m
POS. 37-40  = Quality code
POS. 52-60  = Corrected flow in cubic metres/sec
POS. 62-65  = Quality code
D7H014
Variable 100.00 Surface Water Level
DATE     TIME        COR.LEVEL QUA      COR.FLOW  QUA
19930712 090000          4.450   1        22.082    1
19930716 184200          4.445   1        20.623    1
19930716 223000          4.460   1        25.044    1
19930717 142400          4.467   1        27.173    1
19930717 151200          4.416   1        13.017    1
19930718 114800          4.414   1        12.540    1
19930719 083000          4.422   1        14.499    1
19930719 141200          4.425   1        15.244    1
19930720 155400          4.449   1        21.783    1
19930720 164800          4.467   1        27.173    1
...
```

A.2 Reservoir data sample

```
Data are continuously updated and reviewed.
The format of this file is as follows:
POS.  1-8   = Date of measurement CCYYMMDD
POS. 10-15  = Time of measurement HHMMSS
POS. 27-35  = Corrected level (above spillway) in m
POS. 37-40  = Quality code
POS. 52-60  = Corrected flow in cubic metres/sec
POS. 62-65  = Quality code
D3R003 (FLOW OVER SPILLWAY)
Variable 100.00 Surface Water Level
DATE     TIME       COR.LEVEL QUA    COR.FLOW  QUA
19780415 105400        -0.523   1        0.000   1
19780416 035400        -0.272   1        0.000   1
19780416 061100        -0.221   1        0.000   1
19780416 100500        -0.163   1        0.000   1
19780416 142200        -0.122   1        0.000   1
19780416 213000         0.000   1        0.000   1
19780416 215400         0.008   1        1.271   1
19780417 044800         0.113   1       17.569   1
19780417 070500         0.129   1       20.032   1
19780417 080900         0.136   1       21.109   1
19780417 114700         0.141   1       21.878   1
19780417 232100         0.302   1       46.541   1
19780418 152700         0.538   1      104.788   1
19780419 004200         0.611   1      130.329   1
19780419 090100         0.660   1      149.050   1
19780420 000200         0.724   1      175.412   1
19780420 082900         0.747   1      185.408   1
19780420 161000         0.750   1      186.729   1
19780422 201900         0.943   1      281.959   1
19780423 032200         0.975   1      299.651   1
19780423 145800         0.975   1      299.651   1
19780424 044400         1.029   1      330.738   1
19780424 124000         1.021   1      326.036   1
19780424 232600         1.058   1      346.471   1
19780426 080200         1.110   1      375.249   1
...
```

A.3 Turbine data sample

```
Data are continuously updated and reviewed.
The format of this file is as follows:
POS.  1-8   = Date of measurement CCYYMMDD
POS. 10-15  = Time of measurement HHMMSS
POS. 17-28  = Flow in l/s
POS. 37-40  = Quality code
D3H023
Variable 194.00 Pipeline Discharge
DATE     TIME      FLOW(l/s)        QUA
19760908 070000        0.000          1
19761021 065900        0.000          1
19761021 070000        0.000          1
19761117 095900        0.000          1
19761117 100000    27000.000          1
19761117 105900    27000.000          1
19761117 110000        0.000          1
19761118 115900        0.000          1
19761118 120000    27000.000          1
19761119 065900    27000.000          1
19761119 070000        0.000          1
19761201 095900        0.000          1
19761201 100000    37000.000          1
19761201 100900    37000.000          1
19761201 101000        0.000          1
19761201 104400        0.000          1
19761201 104500    37000.000          1
19761201 115900    37000.000          1
19761201 120000        0.000          1
19761201 121400        0.000          1
19761201 121500    37000.000          1
19761201 122400    37000.000          1
19761201 122500        0.000          1
19761201 143400        0.000          1
19761201 143500    37000.000          1
19761201 163900    37000.000          1
19761201 164000        0.000          1
...
```

A.4 Evaporation data sample

```
Data are continuously updated and reviewed.
The format of this file is as follows:
POS.  1-8   = Date CCYYMMDD
POS. 10-18  = Daily evaporation in mm
POS. 20-24  = Quality code

D7E001
Variable 710.50 Evaporation from A Class Pan

DATE    DAILY EVAP  QUAL
19910501       5.0     1
19910502       9.0     1
19910503       6.0     1
19910504       5.0     1
19910505       6.0     1
19910506       6.0     1
19910507       4.0     1
19910508       5.0     1
19910509       4.0     1
19910510       5.0     1
19910511       5.0     1
19910512       5.0     1
19910513       4.0     1
19910514       3.0     1
19910515       4.0     1
19910516       4.0     1
19910517       5.0     1
19910518       7.0     1
19910519       8.0     1
19910520       5.0     1
19910521       6.0     1
19910522       6.0     1
19910523       5.0     1
19910524       1.0     1
19910525       8.0     1
19910526       4.0     1
...
```